Table of Contents Previous Chapter

1 Introduction

This document specifies a Programming Interface (PI) and corresponding Information Model (IM) which constitute Release 1.0 CAD Framework Initiative Standard for Design Representation data. This document is the work of the CFI Design Information Technical Committee's 1.0 Design Representation Working Group.

1.1 Statement of Purpose

The goal of the CFI Design Information Technical Committee's 1.0 Design Representation Working Group is to define a means for CAD tools to create, access, or modify electronic design data in other CAD tools or databases. This is in support of the CFI goals of tool interoperability, tool interchangeability, data substitution, and data translation.

The Design Representation Working Group's approach to this problem is to define a formal information model (IM) and a programming interface (called the DR-PI) based on the IM. The Information Model identifies the relevant object types in the domain of interest and specifies the attributes of each object type, including its relationships to other objects.

Release 1.0 of the IM and the DR-PI is limited to the domain of ELECTRICAL CONNECTIVITY information. The scope for the IM and the DR-PI is defined by the connectivity information needed for netlisters (EDIF, VHDL, . . .), logic simulators, timing analyzers, synthesis tools, layout tools, and flatteners, all of which work on hierarchical electrical connectivity data.

1.2 Problem Statement

Tools integrated into a CAD framework (or otherwise providing interoperability) require access to design data. A standard means of interactive (run-time) access can avoid translation from one design data format to another. There are two parts to this problem: (1) defining a specific common access mechanism for design data and (2) achieving sufficient consensus across the industry that this specific solution is acceptable and useful. Unless the second part is satisfied, it is unlikely that the CFI DR-PI will become a standard.

So, the 1.0 DR Working Group must do two things:

Define a particular means for CAD tools to represent, access, and manipulate design data, and **Achieve a consensus on that particular means** among the members of the committee in order to drive the formation of a common guideline for the industry. Both are substantial problems, and they must be solved in parallel.

1.3 Solution Approach

An approach has been defined for solving each of the two problems.

1.3.1 The Design Data Access Problem

In the initial meetings of the DR Working Group, the committee quickly decided on an interface-oriented approach for solving the design data access problem. The approach is to define a Programming Interface (PI) which may be implemented in various programming languages (initially in C) and invoked directly from CAD tools. This programming interface is called the Design Representation Programming Interface, or "DR-PI." It was also quickly decided that, to define the DR-PI, an Information Model (IM) for the data to be handled by the DR-PI must first be defined. The IM shows abstractly what design information is made accessible via the DR-PI. The IM represents both the organization and the semantics of that information.

Note that the IM and the DR-PI do not constitute the design of any particular database facility. The IM is not a data model. Instead, each implementor (or provider) of the DR-PI must perform the appropriate mapping between the data model of their actual database and the organization of the CFI Design Representation Information Model. Particular data models and the transformations behind them can (and likely will) be proprietary to particular vendors.

The DR-PI provides access to actual design information, and it also provides an interface to a particular system's data without revealing the underlying implementation. It is a mask which is "put on" by that system and through which all access and manipulation of the data is provided. There is no implication whatsoever that the underlying data must be physically organized exactly like the IM.

To provide this access to the design information, the DR-PI is "handle-" or Object ID- (OID) based. No data structures are created or returned directly. Instead, the DR-PI defines a set of primitive types: such as integer, float, and string, and enumerated types: such as for specification of port direction, and OIDs that provide the "handles" to instances of the entities defined by the Information Model.

OIDs may not be assumed to be pointers and may never be de-referenced directly by DR-PI users. Access to both the attributes and relationships of a design object is only via DR-PI procedures applied to its OID. In particular, the C structure member operators "." and "->" are never used to access fields of a design object because the OID need not be a pointer (though it may be one in a particular implementation).

The Information Model is presented in the form of diagrams using the EXPRESS-G notation, as English text describing those diagrams, and as EXPRESS language text [1]. Each procedure in the DR-PI corresponds to some operation on design data in the domain specified by the IM. When possible, *EXPRESS* is used to define the semantics of those operations. When not, English text is used. Both *EXPRESS* and English are used to state the rules which are to be enforced as the information is manipulated.

1.3.2 The Consensus-Building Problem

Achieving the consensus of a large group of technical people with widely differing personal experience is a complex, laborious task. As the DR Working Group has grappled with this problem, an approach has grown out of the many attempts at listing the issues and bringing them to a state of closure.

This approach is based on democracy and electronic-mail. During meetings, different people will present technical approaches to the problems under consideration. Issues and concerns raised by working group members now must be submitted in writing. Between the meetings, these issues and others are distributed electronically. The issues are presented in a form that allows the members to vote their agreement or disagreement. When disagreement is voted, a member can supply a counter-proposal to be voted on by the group. This approach is proving to be an objective, unemotional procedure for obtaining decisions.

1.4 Scope

The scope of the Information Model is restricted to NETLIST or ELECTRICAL CONNECTIVITY information. This corresponds approximately to the EDIF Netlist View, encompassing Cells, Ports, Nets, Instances, and Port Instances.

Table of Contents Next Chapter

Table of Contents Previous Chapter

2 Definitions

The following definitions are used in this document:

Attributes

An entity is defined in terms of a set of attributes, which represent the entity's essential traits, qualities, or properties. Attributes can contain "primitive" data type values such as strings, integers, reals, and enumerations, or an attribute of one entity can be one or more other entities. *EXPRESS* does not distinguish between an entity's attributes and its relationships as separate notions. Rather, each attribute is also a relationship to the type that represents its value; see [1] in Section 7, "References".

Entity (Type)

An entity represents some thing or concept that is important in a schema. An entity type provides expressions which define the entity. An entity instance is the realization of a specific case of the entity type. Thus while there is an entity type named "CELL," the cell named "Half-Adder" is an Entity Instance of the type CELL. The word entity used alone means entity type. An entity is defined in terms of its data and behavior. The data definition is given as a set of attributes. Behavior is defined in terms of constraints and operations [1].

Entity-Level Model

An Entity-level model is an *EXPRESS-G* model that represents the definitions and their relations, which comprise a single schema [1].

EXPRESS

An information modeling language defined by the *EXPRESS Language Reference Manual* [1]. This document also describes the graphical notation *EXPRESS-G.*

EXPRESS-G

The graphical notation for *EXPRESS*, which is described in Annex D of the *EXPRESS Language Reference Manual* [1].

Objects---Entity Instances

Specific (allocated) cases of entity types, such as a particular "CELL," e.g., "Half-Adder" or "Adder" or "My 64 Bit ALU."

Policy

Rules restricting how entities and their operations may be manipulated.

Programming Interface

A set of procedures or functions used by an application to (in the case of DR) access design data.

Relationships

Attributes of entities that model references to one or more other entities. *EXPRESS* does not distinguish between attributes and relationships explicitly.

Schema

A schema encompasses a set of entity and other declarations which have a related meaning and purpose. A schema can contain other schemas [1].

Schema-Level Model

A schema-level model is an *EXPRESS-G* model which shows the relations between single schemas. It is an abstraction of a model removing much of the detail. Schema-level models should only show schemas, no entities or types.

Session

A session is defined as the interval of time in a program which starts with the "Initiate PI" procedure and ends with the "Quit PI" procedure [2].

Subtype and Supertype

A subtype is a thing which is different from other things of the same general kind. The entity which represents all varieties of that same general kind is called the supertype entity. For any supertype entity ***A*** which has a subtype entity ***B***, the following statement is true:

Every ***B*** is an ***A***, but not every ***A*** is a ***B***.

Type

A type is used to associate an attribute or variable with a data representation. In *EXPRESS*, the type is the kind of an entity or variable. Some types are basic, e.g., integer, real, or string. Other types are more specialized and are defined by the user. Every entity forms a type.

Table of Contents Next Chapter

Table of Contents Previous Chapter

3 The Information Model

One of the primary tasks of the Design Representation Working Group was to define a common Information Model (IM) for design representation. This IM serves as the basis for the Design Representation Programming Interface (DR-PI), which can be used by CAD tools to manipulate design information.

NOTE: While every effort has been made to ensure the accuracy of the IM, if there are any discrepancies between the IM and the DR-PI, the definitions in the DR-PI will take precedence.

3.1 Structure of the Information Model Description

The remainder of this section provides a detailed description of an Information Model for representing electronic circuits. Ultimately, most aspects of circuit structure (i.e., schematics, layouts, and netlists) will be represented. However, at present, hierarchical netlist connectivity is the single aspect covered by this model.

In an effort to manage the definition process, the IM has been partitioned. Each major portion of the model is described in a separate section using textual descriptions and information model diagrams. The most primitive or low-level concepts are described first, followed by descriptions of additional entities and types needed to build up to the final model.

As mentioned previously, the information model described in this document will be constructed in the *EXPRESS* language [1]. One of the primary goals for using *EXPRESS* is to capture the information model explicitly in a computer readable format. *EXPRESS* provides a syntax for defining data types, data entities, the behavior among entities, and inheritance among the definition of entities. This description will include "fragments" of the *EXPRESS* text corresponding to the concept being described. These fragments do not represent the entire model. For example, this section does not present the definition of the function **unique_names()**. Refer to Appendix A for the entire *EXPRESS* text description of the DR connectivity model including any functions used, such as **unique_names()**. Refer to [1] for a description of the *EXPRESS* language. All of the diagrams in this document are drawn using the *EXPRESS-G* graphical notation which is briefly explained in the next section. Refer to [1] also for a detailed description of *EXPRESS-G*.

The first diagram is shown in Figure 3.1, "Base Object Model Diagram," and is described in Section 3.3. The base object model captures the top of the entity hierarchy used to describe the information model.

The second diagram, shown in Figure 3.2, "Base Connectivity Model Diagram." This model represents a high-level abstraction of the basic connectivity model for electronic circuits.

3.2 Description of *EXPRESS-G*

EXPRESS is a language used to model information. The language is described in the ISO document, *EXPRESS Language Reference Manual*, TC184/SC4 WG5 Document N14 [1]. *EXPRESS* can be used to

model:

- Information units that package data elements,
- Constraints on data elements or the information units as a whole, and
- Operations on those information units [1].

All of the information models presented in this document will be described formally in the *EXPRESS* language (refer to Appendix A, "Information Model").

EXPRESS is a language. Hence, information models written in *EXPRESS* are textual. However, a graphical notation has also been defined, dubbed *EXPRESS-G*. This notation is described in detail in Annex D of the document cited above. However, since *EXPRESS-G* is a new notation for many readers, a brief description is provided here.

EXPRESS-G is a graphical notation for the display of information models. The notation only supports a subset of the *EXPRESS* language, and therefore *EXPRESS-G* models are normally abstractions of *EXPRESS* models [1]. There are equivalent notations for bit-mapped graphics and ASCII character or "typed" diagrams. *EXPRESS-G* has symbols to represent *EXPRESS* entities, user-defined types, base types, relationships (required and optional), and inheritance. Table 3.1, "Express-G Legend", provides a legend for understanding *EXPRESS-G* diagrams.

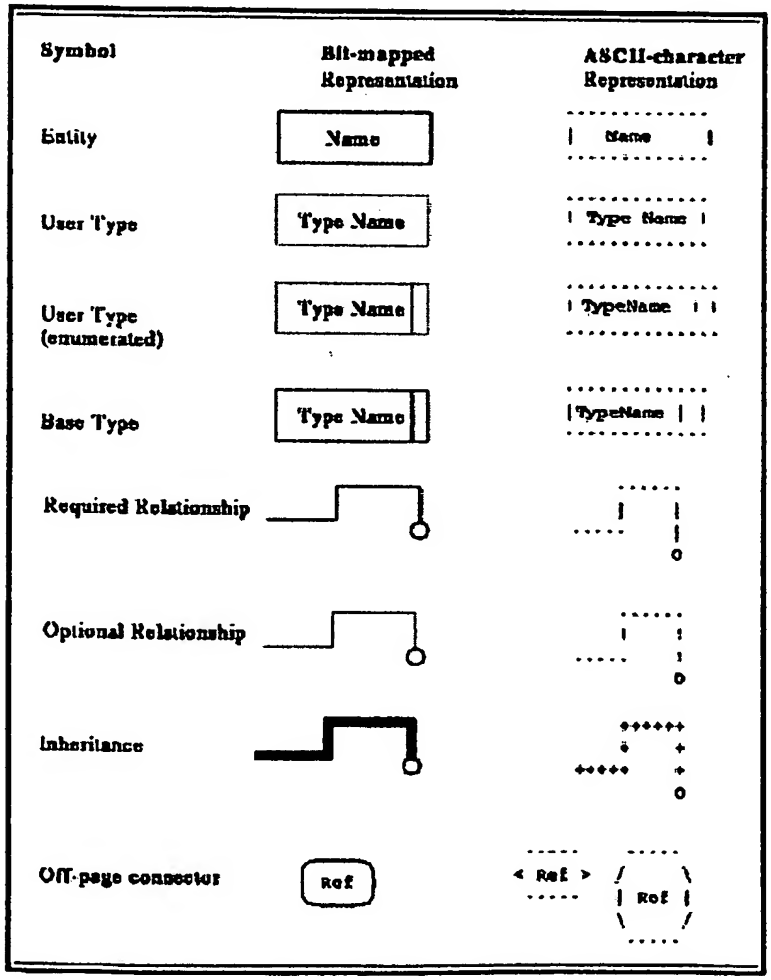

Table 3.1 EXPRESS-G Legend

The small circle on the end of relationship arcs indicates direction only; it has nothing to do with cardinality. The cardinality of a relationship will be annotated on the relationship using text. Refer to the *EXPRESS* manual [1] for more informati on about *EXPRESS-G*.

3.3 The Base Object Model

The entire Design Representation Information Model is derived from a single entity which models the basic, low-level behavior of any entity in the model. This low-level behavior is intended to capture the essence of the notion of an object. The basic behavior of all objects in the DR model is that they will be typed and may have an optional list of properties associated with them. Additionally, many objects will have a name. The EXPRESS-G diagram for this basic object model is shown below in Figure 3.1 as an entity-level model. Following the figure is a more-detailed description of the entities and types depicted in the diagram.

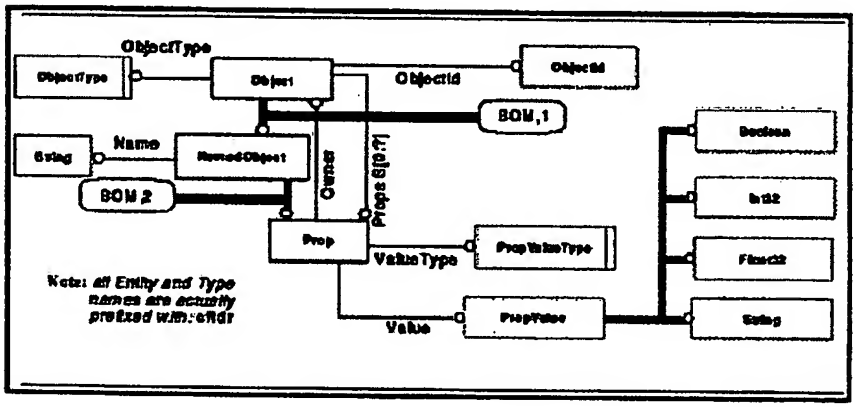

Figure 3.1 Base Object Model Diagram

3.3.1 Type: ObjectType

```
TYPE cfidrObjectTypeT = ENUMERATION OF (

   CFIDR_CELL,

   CFIDR_INST,

   CFIDR_LIB,

   CFIDR_NETBUNDLE,

   CFIDR_NETBUS,

   CFIDR_NETSCALAR,

   CFIDR_PORTBUNDLE,

   CFIDR_PORTBUS,

   CFIDR_PORTINSTBUNDLE,
```

```
    CFIDR_PORTINSTBUS,

    CFIDR_PORTINSTSCALAR,

    CFIDR_PORTSCALAR,

    CFIDR_PROP,

    CFIDR_ENCAPSULATEDVIEW,

    CFIDR_NETLISTVIEW

    );

END_TYPE;
```

DESCRIPTION

ObjectType defines all possible object types. Values of type cfidrObjectTypeT will be used as the value of the *ObjectType* attribute for all DR objects. These types should not be confused with the base object types defined by EXPRESS which include r eal, integer, string, boolean, and logical.

RATIONALE

All entities shall be typed. A fixed set of object types is needed to provide a common basis for the information in a design description which can be shared. Note that only the non-abstract entity types are included in this list since they are the only ty pes from which entity instances can be created.

3.3.2 Type: ObjectId

```
TYPE cfidrObjectIdT = INTEGER; END_TYPE;
TYPE  cfidrCellIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrEncapsulatedViewIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrInstIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrLibIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrNamedObjectIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrNetIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrNetBundleIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrNetBusIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrNetScalarIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrNetlistViewIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrPortBundleIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrPortBusIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrPortIdT = cfidrObjectIdT; END_TYPE;
```

```
TYPE  cfidrPortInstIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrPortInstBundleIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrPortInstBusIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrPortInstScalarIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrPortScalarIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrPropIdT = cfidrObjectIdT; END_TYPE;

TYPE  cfidrViewIdT = cfidrObjectIdT; END_TYPE;
```

DESCRIPTION

Programming Interfaces (PIs) will refer to objects by object identifiers (OIDs) which cannot be assumed to be C pointers or any other specific language data type. They will differ in different implementations because OIDs in any given system can be differ ent kinds of objects, and vendors need to be able to make changes to the definition of OIDs without forcing program structures to change. The EXPRESS code declares the cfidrObjectIdT as INTEGER because it must use some Express primitive type. This does n ot mean an implementation must use an INTEGER.

NOTE: The Architecture Working Group has decreed that all machines of the same processor architecture have a common cfidr.h header file and that the Object IDs should be twice the size of void * (i.e., twice the size of an address).

RATIONALE

Since all objects are typed and all objects are accessed via OIDs, an ID entity is required. The intent is that IDs are the "handles" through which objects are manipulated. An application tool should not be required to possess knowledge of the implementat ion of IDs to use them. They are non-transparent handles.

CONSTRAINT

IDs are valid for the current session only. Hence, IDs cannot be stored in a persistent database.

In order to minimize a tool's need to explicitly test the type of an object, each type of object has a specific type for its ID. Therefore a tool usually should only need to determine whether a particular ID is valid.

3.3.3 Entity: Object

```
ENTITY cfidrObject

  ABSTRACT SUPERTYPE OF (ONEOF

 (cfidrNamedObject, cfidrPortInst));

  ObjectType:  cfidrObjectTypeT;

  ObjectId:    cfidrObjectIdT;

  Props:       SET [0:?] OF cfidrProp;

WHERE unique_names(Props);
```

```
END_ENTITY;
```

DESCRIPTION
The entity Object will be a supertype for every other entity defined in this information model. It models the basic behavior of all objects: all objects are typed; all objects have a unique identifier (an ID); and all objects may have properties associate d with them.

RATIONALE
The DR Working Group intends this information model to be designed in an object-oriented fashion. In particular, the model will use inheritance so that common aspects of otherwise distinguished objects will be specified in a common root object. The entity Object is the root of the entire inheritance graph. Since EXPRESS supports the modeling of multiple inheritance, this need not be the only root entity type. The intent is that this entity provide the minimal attributes that any (cfidr) object needs to be considered a valid member of the object community.

CONSTRAINT
The Name attribute of properties must be unique within the scope of a single object. That is, no two properties associated with one object may have identical names.

3.3.4 Entity: NamedObject

```
ENTITY cfidrNamedObject

  SUBTYPE OF (cfidrObject)

  ABSTRACT SUPERTYPE OF (ONEOF(cfidrProp, cfidrLib,

                cfidrCell, cfidrView,

                cfidrPort, cfidrNet, cfidrInst));

  Name:  cfidrStringT;

END_ENTITY;
```

DESCRIPTION
As with Object, NamedObject defines an entity which will serve a supertype for many other objects in the model. Any object which has a text name string will be defined as a subtype of NamedObject.

RATIONALE
Using inheritance, the ability for an object to be named is modeled by making that object a subtype of NamedObject. It is expected that many objects in the model will be named. Hence inheritance is used to ensure that a consistent notion of naming is defined and used for each object requiring a Name attribute.

3.3.5 Entity: Prop

```
ENTITY cfidrProp

  SUBTYPE OF (cfidrNamedObject);

  ValueType: cfidrValueTypeT;
```

```
  PropValue: cfidrPropValue;

  Owner:     cfidrObject;

END_ENTITY;
```

DESCRIPTION
A Prop allows an application to effectively extend the database schema by including its own Object attributes as properties. A Prop consists of a name/value pair and can be associated with any object. Prop values are typed (see ValueType below).

RATIONALE
No information model can explicitly model every need of every possible application tool. Hence, the model must be inherently extensible to support additional information needs. The intent of the property entity is to partially provide this extensibility. Properties may be used to annotate other entities with additional information that is not formally part of the information model.

CONSTRAINT
When a Prop is removed, any Props associated with it are also removed.

3.3.6 Type: ValueType

```
TYPE cfidrValueTypeT = ENUMERATION OF

  (CFIDR_UNDEFINED_VALUETYPE, CFIDR_INT32, CFIDR_STRING,

   CFIDR_FLOAT32, CFIDR_BOOLEAN);

END_TYPE;
TYPE cfidrPropValue = SELECT

 (cfidrBooleanT, cfidrInt32T, cfidrFloat32T, cfidrStringT);

END_TYPE;
```

DESCRIPTION
These type definitions describe the valid types for values of the *Value* attribute of a Prop entity. The SELECT type PropValue allows the PropValue attribute of Prop to be any the four shown types.

RATIONALE
A Prop's value must be typed to provide a common model for properties. This may limit the extensibility of the model. To allow user-definable property types is beyond the scope of the Information Model at this stage.

3.4 Overview of the Base Connectivity Model (BCM)

This section describes the portion of the model that represents hierarchical netlist connectivity. The Base Connectivity Model (BCM) is shown in Figure 3.2.

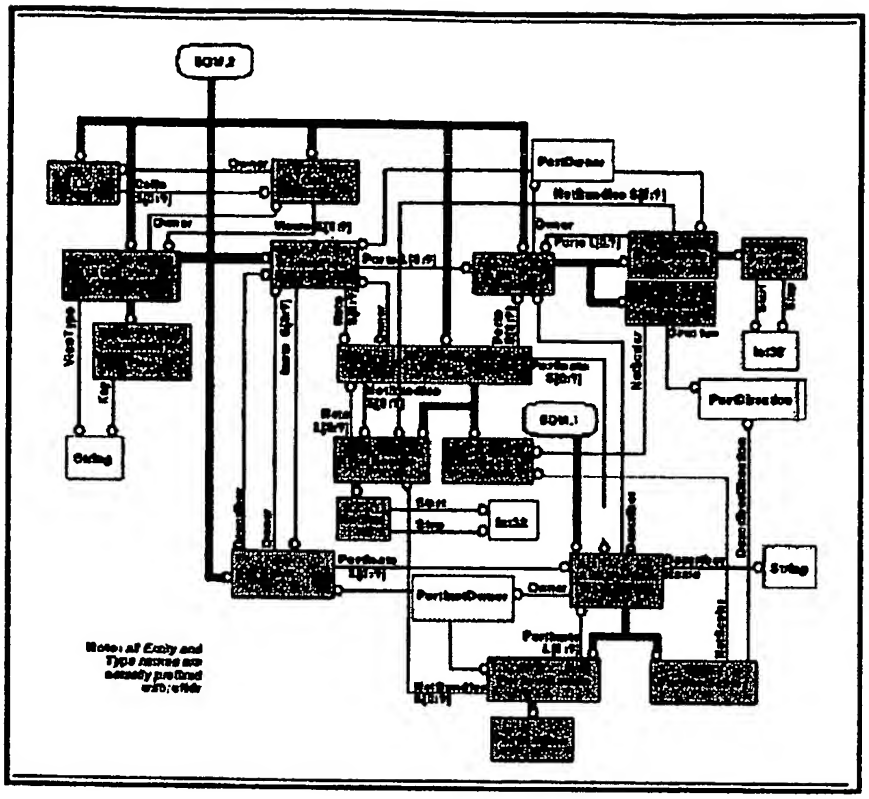

Figure 3.2 Base Connectivity Model Diagram

This model represents the objects and relationships used to represent hierarchical netlists with bundles in CFI 1.0.0 Design Representation.

In Figure 3.2 there is no modeling significance to the gray entities, they are just to highlight the "main" entities in the model.

The following two sections describe the model shown in Figure 3.2. First, a conceptual description is given for the technique of representing connectivity using the model. Following that is a detailed description of each entity and type in the BCM diagr am and the constraints on their behavior.

3.5 Base Connectivity Model Tutorial Description

Although the probability is high that anyone reading this document has knowledge of hierarchical circuit connectivity and is familiar with at least one system for representing it, there is a reason for presenting an overview in tutorial fashion here. Specifically: the model asserts throughout rules about how entities in it are to be combined to represent a hierarchical netlist. This tutorial describes the entire model and all the rules in one place. Each of the rules will be stated again as constraints on the entities defined later in this section.

Hierarchical design supports the notion of building up the behavior of a design by collecting and connecting together other designs. Each of the other designs can in turn be built from yet other designs, etc., recursively until a subdesign is reached which is composed of primitive design elements. In this case a primitive design element is one for which no further interior structure is known. Primitive design

elements are also referred to as leaf cells.

A bottom-up example is to use a transistor as a primitive element or leaf cell. Transistors then are used to build and-gate, nand-gates, inverters, etc. These gates are used to build an exclusive-or-gate, which is used with other gates to build an half-ad der cell, which is used with other cells to build an adder cell, which is used with other cells to build an ALU, etc.

In order to represent a hierarchical design, there must be a way to represent the design under consideration and the fact that it is composed of other designs which are connected in some way. In the model described herein, a particular implementation of a design is represented by the NetlistView entity. The inclusion of smaller designs of which a NetlistView is composed are represented by the Inst entity. The term Inst is an abbreviation for the word Instance.

Instantiation is the act of using one design in the structure of another design. In this model, the design which is being used is the Inst, and the design in which an Inst is used is the NetlistView. As an analogy, think of the NetlistView as a printed circuit board and an Inst as a chip which is soldered down onto the circuit board. A NetlistView may have more than one Inst within it and multiple Insts of a single design may be placed into a NetlistView. An Inst represents the use of one design within an other design. Each Inst represents an instantiation of one NetlistView within another NetlistView.

An instantiation is not exactly a copy nor is it exactly a symbolic reference. For example, in a NetlistView named "Half-Adder" an Inst of a NetlistView named "XOR" represents the fact that the design of an xor gate is used as a component in the design of a half-adder.

Having provided a representation for the hierarchical structure of designs using the NetlistView and Inst entities, it is necessary next to provide a representation for interconnecting the Insts within a NetlistView.

The connectors of a design are represented by the Port entity. A Port has a name and a direction. The direction of the Port defines how signal information flows through the Port. A signal either flows into the design or out from the design through the Port. Or, in some cases, a Port is bidirectional and can conduct signal information both into and out of a design.

The pins of an Inst are represented by the PortInst entity. Since an Inst represents the use of a specific NetlistView, the PortInsts belonging to an Inst correspond exactly to the Ports on the NetlistView referenced by the Inst. For example, if an Inst "xor1" represents the use of an XOR gate within a design then each of that Inst's PortInst entities will correspond to a single Port entity in the NetlistView of the XOR gate that was used to create this Inst. The correspondence between an Inst and the Net listView it represents is referred to as the "Describer Relationship". An Inst is completely "described" by the NetlistView it represents. This same corresponcence exists between a PortInst and the Port it represents. Both the Inst and PortInst entitie s contain a Describer attribute.

The fact that PortInst's attributes for name and direction are *DescriberName* and *DescriberDirection* indicate the close tie required between each PortInst and its *Describer* Port. See Section 6, "View Selection", on how this very tight relationship between Insts/PortInsts and their *Describers/Ports* can be relaxed. View Selection provides a mechanism to specify rules for choosing an alternative (substitutable) NetlistView (and its correspo nding Ports) for any given Inst.

To complete the initial model, the Net entity is used to represent each set of connections between PortInsts and Ports within a NetlistView. When a collection of PortInst and Port entities are associated

with a Net, it reflects the intention that each PortInst and Port in the collection will have exactly the same signal information at all times. A Net may connect only PortInsts, only Ports, or a mixture of the two.

One final concept is that of bundles. In many scenarios, it is convenient for a designer to group a set of signals together and refer to the set as single signal. An example of this is the RS-232 standard signal set for the interconnection of serial communication equipment. The RS-232 standard defines the set of signals expected to be found in the 25 pin connectors common on most telephone modems used to interconnect computer systems. An RS-232 cable has multiple wires which are bundled together into a single cable which can be connected once. In electronic design, there are times when it is convenient to refer to the RS-232 signal collection as a single signal, and there are times when it is necessary to refer to each individual signal separately. The same is true with respect to the connectors for RS-232 signals. The RS-232 connector bundles together 25 individual connectors.

In the BCM, this concept is represented by introducing the notions of ***Scalar***, ***Bundle***, and ***Bus***. A Scalar is an individual thing which may not be unbundled into anything else. A Bundle is an ordered collection and a Bus is an ordered collection with index values for each position. The idea of Scalar, Bundle, and Bus is applied to Nets, Ports, and PortInsts.

A Net may be either a NetScalar, a NetBundle, or a NetBus. In a sense, there are now 3 "types" of Net entities. A NetScalar entity represents one individual signal which may not be further decomposed into subsignals. A NetBundle entity is a collection of Nets, each of which is optionally a NetScalar (with no further substructure), a NetBundle, or a NetBus. A NetBus entity is a NetBundle with two additional attributes of *Start* and *Step* which define the range of *Index* values associate d with the *Positions* in the bundle.

Thus NetBundles have only the *Names* for each Net and an implicit *Position* in the bundle for each Net. NetBusses are NetBundles that also have an *Index* value for each position. This *Index* is restricted to be monotonically changing from position to position by a fixed integer *Step* (positive or negative) but may *Start* at any integer value.

NetBundles do not hide the *Names* of their member nets. All Nets in any one NetlistView are required to have unique names. Nets can appear in more than one position in a given bundle and in more than one NetBundle. A given name for a Net in a part icular NetlistView always refers to the exact same Net.

A similar structure exists for Ports and PortInsts, resulting in the definition of **PortScalars**, **PortBundles**, **PortBusses**, **PortInstScalars**, **PortInstBundles** and **PortInstBusses**. The PortInstBundles (and PortInstBusses) get their structure entirely from the corresponding PortBundles (and PortBusses).

However, a significant difference from NetBundles is that PortBundles hide the Names of their members from other PortBundle contents (and from the names of the Ports that are directly in the NetlistView). Thus Port *Names* may be reused without refe rring to the same object. The other difference is that a PortBundle (and thus also a PortInstBundle) cannot repeat a member in two different positions, therefore any one name only appears one time in a given bundle.

The rules covering the interconnection of Ports and PortInsts using Nets are complicated. To help clarify the explanation, the phrase "in-a-bundle" will be appended to the terms NetScalar, PortScalar, and PortInstScalar to distinguish items contained by a bundle from items NOT contained by a bundle.

As explained prior to introducing the notion of a bundle, a Net represents a connection between Ports and/or PortInsts. So it follows that a NetBundle will represent a connection between collections of Ports and PortInsts. The following rules summarize t he bundle connectivity model:

Allowed Connections:

A NetScalar connects to [0:?] (read as zero or more) Port[inst]Scalars;
A Port[Inst]Scalar is connected to [0:1] (read as zero or one) NetScalar(s).
When a Net that is in one (or more) NetBundle(s) connects to a Port[inst] that is in a Port[Inst]Bundle, this implies a connection between *all* NetBundles containing the Net to the Port[Inst]Bundle containing the Port.
A NetBundle connects to [0:?] Port[Inst]Bundles.
A Port[Inst]Bundle is connected by [0:?] NetBundles.

Disallowed Connections:

A NetScalar connects to NO Port[Inst]Bundles directly. Port[Inst]Bundles are NOT directly connected by a NetScalar.
A NetBundle connects NO Port[Inst]Scalars directly;
A Port[Inst]Scalar is NOT directly connected by a NetBundle.

Shorted Ports

The connection of multiple PortScalars to one NetScalar creates a short between the PortScalars. This also means that the NetScalars connected to the PortInstScalars which are instantiations of the shorted PortScalars are electrically equivalent. Thus, th eir names are aliases of each other.

NetBundle-PortBundle Asymmetry

Why are bundled Nets and Ports treated differently? Many people initially expect them to be symmetric, since a Port is conceptually just a special Portion of a Net. But there are two reasons for this lack of symmetry:
(1) cell content descriptions are substantially *simplified*, and
(2) the cascading *delete* requirements are different for Nets and Ports.

These are each explained in more detail below:

(1) Simplified Logical Connectivity Descriptions
The *simplification* comes from the elimination of complex facilities to represent the ripping of NetBundles to connect their constituents to other NetBundles or scalars. Instead, the default basis for Net connectivity within a cell is by unique Net name within the cell. That is, any scalar or bundle named X in a cell is the same scalar or bundle (connected to be electrically the same), where X may be any name. In fact, since they are the same, we no longer even think in terms of connectivity. All ap pearances in a cell of a Net (Scalar, Bundle, or Bus) named X are simply the same scalar, bundle, or bus.

This principle applies to each NetlistView of a cell. Each NetlistView has its own separate Net name scope. However, NetScalars whose *IsGlobal* attribute is true are global in name scope such that all appearances of the same name on any global NetSc alar represent the same NetScalar. That is, all global NetScalars with the same name carry the same electrical signal.

Thus, while Nets may be "connected" by name inside a view, Net to Port connections cannot be by name. Since Port names are available outside the view, they are likely to be different as external needs require. Net to Port connections therefore must be exp licitly expressed, and not left to default. Therefore Net to Port or Net to PortInstance connections must be explicitly expressed or connected by the matching order of bundle constituents.

(2) Cascading Deletion Semantics
Secondly, the deletion semantics for NetBundles and PortBundles are different. The deletion of a NetBundle must not cascade to delete the scalars within the NetBundle. NetScalars are the essence of the design, and NetBundles are just an artifact for conve nience. Conversely, the deletion of a PortBundle is the removal of that Port and all its constituents from the external appearance of the view. Thus, deletion of a PortBundle cascades to also delete all the constituent Ports within that PortBundle.

3.6 Base Connectivity Model Detailed Description

In this section, each entity in the BCM is described in detail. The content of the information for each entity is described along with any constraints on the behavior of the entity.

3.6.1 Entity: Lib

```
ENTITY cfidrLib

  SUBTYPE OF (cfidrNamedObject);


  Cells:     SET [0:?] OF cfidrCell;


  WHERE

  unique_names(Cells);

END_ENTITY;
```

DESCRIPTION
Lib is an abbreviation for Library. The Lib entity is intended to collect together all of the design data that shares a common set of characteristics. A Lib will contain a set of Cell entities on the *Cells* attribute. Each Cell represents a specific component of a design. The realization of a specific view type for a specific Cell is modeled by theView entity (described later). A Lib has no *Owner* attribute.

The *Name* attribute for a Lib (inherited from NamedObject) is a single string which labels the Lib.

RATIONALE
The intent of the Lib entity is to model the well-known concept of a cell library. At minimum, a library will have a collection of cells with views of different types. The EDIF notion of a library should map to this concept in time (when technology inform ation is added). Since a Lib is an object, it may contain properties. Using properties, additional information may be loaded onto a Lib to support specific needs. To clarify the meaning of

these entities, consider the following examples:

- Lib: a standard cell one micron component library or a library containing the entire definition of a microprocessor.
- Cell: a Nand gate, an ALU, or a printed circuit board within a system.
- View: the layout for a Nand gate or the schematic symbol for an ALU.

CONSTRAINT
A Lib's *Name* attribute must be unique with respect to all other lib names known by the current PI session.

When a Lib is removed from a session (*purged* or *destroyed*), all the Cell and Prop objects contained by the Lib become inaccessible in the session.
All Libs accessible during a session must have unique names.

3.6.2 Entity: Cell

```
ENTITY cfidrCell

  SUBTYPE OF (cfidrNamedObject);


  Owner:      cfidrLib;

  Views:      SET [0:?] OF cfidrView;


  WHERE

  unique_names(Views);

END_ENTITY;
```

DESCRIPTION
A Cell represents an abstract portion of a design that may be implemented with several Views. A Cell is owned by a Lib via the *Owner* attribute. To model the realization of multiple, specific implementations, a Cell provides a set of View entities t hrough the *Views* attribute.

RATIONALE
The generic notion of a design cell is intended here. A Cell is a specific component of a design such as an "Adder" or "ALU." However, the Cell entity does not imply a specific implementation of the component. Specific implementations of a Cell are realiz ed in the View entities (described later).

CONSTRAINT
A Cell's *Name* attribute must be unique with respect to the names of all Cells in the set of *Cells* in the *Owner* Lib.

Purging or destroying a Cell within a session causes any View and Prop entities owned by the Cell to become inaccessible.

3.6.3 Entity: View

```
ENTITY cfidrView
```

```
  SUBTYPE OF (cfidrNamedObject)

  ABSTRACT SUPERTYPE OF (ONEOF

  (cfidrEncapsulatedView, cfidrNetlistView));


  Owner: cfidrCell;

  ViewType: cfidrStringT;


END_ENTITY;
```

DESCRIPTION
A View entity represents an abstraction of the design. It represents a specific implementation of one particular Cell. As such, a View is owned by exactly one Cell via the *Owner* attribute.

View is an abstract type. Specific views are either an EncapsulatedView, representing a type that is unknown to the DR-PI, or a NetlistView containing electrical connectivity information that is accessible through the DR-PI. Both types of View may also h ave their type further classified by the *ViewType* attribute string.

RATIONALE
In future versions of the model, the View entity will be further refined into additional sub-entities to partition the behavior and information associated with different aspects of Views (e.g., topology versus topography).

CONSTRAINT
A View's *Name* attribute must be unique with respect to all View names within the scope of a Cell.

3.6.4 Entity: EncapsulatedView

```
ENTITY cfidrEncapsulatedView


  SUBTYPE OF (cfidrView);


  Key: cfidrStringT;


END_ENTITY;
```

DESCRIPTION
An EncapsulatedView represents a view of the cell about which very little is known to the DR-PI. It is a place to "hook" outside information such as a file containing behavioral

information.

The *Key* attribute is expected to typically be a file name or other such thing that allows a system to locate the actual information associated with this view.

EncapsulatedViews may also have their type further classified by the *ViewType* attribute string.

RATIONALE
EncapsulatedView provides a standard way to associate outside information with a Cell without resorting to the intrinsically non-standard use of properties.

NOTE: No mechanism is provided to access ports or other aspects within an EncapsulatedView. An EncapsulatedView must either already have internally-named Ports with the same names as used in the other Views, or some tool-specific mechanism must be u sed to relate between them.

3.6.5 Entity: NetlistView

```
ENTITY cfidrNetlistView


  SUBTYPE OF (cfidrView);


  Ports:    LIST[0:?] OF cfidrPort;

  Nets:     SET [0:?] OF cfidrNet;

  Insts:    SET [0:?] OF cfidrInst;


  WHERE

  unique_names(Insts);

  unique_names(Nets);

  unique_names(Ports);


END_ENTITY;
```

DESCRIPTION
A NetlistView entity represents an abstraction of the design that uses instances of other NetlistViews and may itself be in turn instantiated in another NetlistView to build a design hierarchy. Each NetlistView represents the electrical connectivity of a specific implementation of the *Owner* Cell.

The NetlistView entity models the structure and electrical connectivity of a hierarchical netlist. It is similar in scope to the EDIF 2 0 0 NETLIST [5] view. Electrical connectivity refers here to the logical or electrical connections that carry logical o r electrical signals within and between the cells in the

design.

Ports represent the "places" on the NetlistView where connections can be made to instances of this view. They provide access to the signals inside this view.

Insts represent the hierarchical references to (or inclusion of) other NetlistViews in the definition of this NetlistView.

Nets represent the signal carrying connections between the *Ports* of this NetlistView and *PortInsts*. *PortInsts* are located on *Insts* and correspond to the *Ports* of the NetlistViews that describe each *Inst*. ***Port, Net, Inst,* and *PortInst* are each described further in the sections on those entities.**

> **RATIONALE**
> **A Cell may have more than one View including more than one NetlistView. Each View corresponds to some specific aspect of or alternative for implementing a Cell.**
>
> **The NetlistView models the essence of an electrical circuit netlist. It is very similar to the basic information in the EDIF 2 0 0 NETLIST [5] view.**
>
> **The NetlistView entity represents a key concept in this model. It embodies the concept of a piece of design that is defined by its interface Ports and optionally by instantiation and connection of other NetlistViews. This design piece itself can, in turn, be used (multiple times) in the definition of other Cell's NetlistViews.**

In future versions of the model, the NetlistView entity may be further refined into sub-types to distinguish between "contents" views with substructure (nets and insts) and "interface" views without substructure (ports [and maybe nets] only). Another dist inction can be made (as in the new EDIF 3 0 0) between logical connectivity and the structure used in implementing it with specific objects such as the graphics in a schematic or a symbolic layout view. For instance, additional structure is imposed on th e connectivity in a schematic by junctions, rippers, and off-page connectors.

CONSTRAINT
Each Port's *Name* attribute must be unique with respect to the names of all other Ports in the list of *Ports*.

Each Inst's *Name* attribute must be unique with respect to the names of all other Insts in the set of *Insts*.

Each Net's *Name* attribute must be unique with respect to the names of all other Nets on the set of *Nets*.

3.6.6 Entity: Inst

```
ENTITY cfidrInst

  SUBTYPE OF (cfidrNamedObject);


  Owner:        cfidrNetlistView;

  Describer:    cfidrNetlistView;

  PortInsts:    LIST [0:?] OF cfidrPortInst;

END_ENTITY;
```

DESCRIPTION

The Inst entity models the hierarchical usage of a NetlistView as a part of another NetlistView. The NetlistView which describes the Inst is in the *Describer* attribute. Insts also have PortInsts corresponding one-to-one with the Ports of the instan tiated NetlistView. These are in the list that is the *PortInsts* attribute. The containing NetlistView which is using this Inst is in the *Owner* attribute.

RATIONALE

Electronic design often creates new components by connecting existing components. The outcome is intrinsically hierarchical. To represent a hierarchy of connected components, the notion of instantiation is used. An existing component is placed within a ne w component by instantiation. The Inst entity realizes the instantiation of a component. It is not exactly a copy nor is it exactly a reference. It is both at the same time. An Inst is intended to represent the usage of a component without duplicating the definition of the component each place it is used. Rather, the Inst provides an indirect reference to the definition of the instantiated component while also providing a location for information specific to the usage of the component at the point of inst antiation.

CONSTRAINT

A NetlistView may not instantiate itself either *directly* or *indirectly*. In other words, none of the instances in a view may be of that same view nor may they be of views which have instances of that same view or instances of views which have instances of that same view, etc. Simply put: no recursive instantiation is allowed. This however does not preclude instantiating another view of the same cell—something that might be done to collect views into pages, for instance.

NOTE: The Programming Interface is only required to check for *direct* recursion because of the computational cost associated with checking for arbitrary *indirect* recursion.

3.6.7 Entity: Port

```
TYPE

  cfidrPortOwner = SELECT

    (cfidrNetlistView, cfidrPortBundle);

END_TYPE;
ENTITY cfidrPort

  SUBTYPE OF (cfidrNamedObject)

  ABSTRACT SUPERTYPE OF ONEOF

    (cfidrPortScalar, cfidrPortBundle);


  Owner:       cfidrPortOwner;

DERIVED

  Position: cfidrInt32T find_position_in_owner();

END_ENTITY;
```

DESCRIPTION

Ports are the means of communicating information between NetlistViews in a design hierarchy. Ports are the external interface for a NetlistView. They correspond to PortInst entities when a NetlistView is instantiated.

Port is an abstract type. All actual ports are either PortScalar, PortBundle, or PortBus.

Internal to a NetlistView, a Port is connected to a Net, thus passing along information into this cell view and then to lower levels of the design hierarchy.

A Port is owned either by a PortBundle or a NetlistView entity via the *Owner* attribute (which necessitates the introduction of the cfidrPortOwner SELECT TYPE). A NetlistView may have many Ports.

The *Position* of a Port is the location of this particular port in the list of *Ports* in *Owner*. The initial or "leftmost" position is numbered 0 and each subsequent port has position equal to one more than that of the port to its "left ". Thus for a list with three ports (A, B, C), A has position 0 and C has position 2.

NOTE: Ports have position regardless of whether the *Owner* is a NetListView or a PortBundle; in both cases they are on a list of Ports.

RATIONALE

A Port represents visibility for any Net of a NetlistView such that, when that NetlistView is used as an Inst in another "higher" NetListView, the Net can be connected via the corresponding PortInst to a Net in the "higher" NetlistView. In a sense, placin g a Port on a Net is declaring to the world that the Net is connectable from above in the hierarchy. It provides a "port of entry" to or an exit from the NetlistView.

CONSTRAINT

The *Name* attribute of a Port must be unique with respect to the *Name* attribute of every other Port on the list of *Ports* in the *Owner* (NetlistView or PortBundle).

3.6.8 Entity: PortScalar

```
ENTITY cfidrPortScalar

  SUBTYPE OF (cfidrPort);


  Direction:  cfidrPortDirection;

  NetScalar: OPTIONAL cfidrNetScalar;

END_ENTITY;
```

DESCRIPTION

A PortScalar is a subtype of the abstract Port entity. A PortScalar is a single connection point to signal information which may not be further divided into sub-parts. The direction of the PortScalar is specified with the *Direction* attribute. A Por t's direction is specified by the PortDirection enumeration. A PortScalar is directly connected to a NetScalar via its *NetScalar* attribute.

RATIONALE
A PortScalar is a "leaf" signal carrier. It provides a connection point to a single wire or signal carrier.

Only PortScalar has a direction. PortBundles and PortBusses have no overall direction.

3.6.9 Type: PortDirection

```
TYPE cfidrPortDirection = ENUMERATION OF
       (CFIDR_UNDEFINED_PORTDIRECTION,
        CFIDR_INPUT,
        CFIDR_OUTPUT,
        CFIDR_IO);
END_TYPE;
```

DESCRIPTION
This type defines the legal values for the *Direction* attribute of a PortScalar entity as well as for the derived attribute *DescriberDirection* for the PortInstScalar entity.

RATIONALE
Port direction must be consistently specified to achieve an interoperable Design Representation model.

3.6.10 Entity: PortBundle

```
ENTITY cfidrPortBundle
  SUBTYPE OF (cfidrPort)
  SUPERTYPE OF (cfidrPortBus);

  Ports:       LIST [0:?] OF UNIQUE cfidrPort;
  NetBundles:   SET [0:?] OF cfidrNetBundle;
DERIVED
  Size:  SIZE_OF(Ports);
END_ENTITY;
```

DESCRIPTION
A PortBundle entity models the ability to group many Ports together and treat the group as a single entity under appropriate conditions. PortBundles have substructure. That is, a PortBundle contains other Port entities which are held in the *Ports* a ttribute. A PortBundle may be made up of PortScalars or other PortBundles hierarchically.

A PortBundle is connected to 0 or more NetBundles via its NetBundles attribute.

RATIONALE

The tutorial explanation for the BCM (Figure 3.2 Base Connectivity Model Diagram on page 13) contains a statement of the rationale for PortBundles. In general, PortBundles allow a group of PortScalars and other PortBu ndles to be treated as a single entity when it is appropriate and convenient to do so.

CONSTRAINT

PortBundles may not recursively contain themselves either directly or indirectly. The PI is required to check for direct recursion at PortBundle creation time but not to check for indirect recursion at that time due to the potential computational cost.

3.6.11 Entity: PortBus

```
ENTITY cfidrPortBus

  SUBTYPE OF (cfidrPortBundle);

  Start: cfidrInt32T;

  Step:  cfidrInt32T;

END_ENTITY;
```

DESCRIPTION

A PortBus entity is simply a PortBundle with additional attributes that associate a specific index value with each position in the bundle.

***Start* is the index value for *Position* 3.**
***Step* is the difference in index values of *Position* n and n+1.**

RATIONALE

Many different terminologies were discussed in the DR TSC for non-scalar or composite signals: bundles, arrays, groups, and busses were the most common terms. In most cases, regardless of name, the Bundle entity, an ordered grouping of items (unique in the case of Ports), has the correct semantics while all other terminology distinctions are based mostly on naming conventions.

The only semantic distinction on which agreement could be reached was that, in addition to the characteristic of being an ordered group, sometimes these groups also had an index to identify each position, that the index usually changed monotonically in v alue, and that the index of position 0 might be any value. The term Bus is used for these groups in which the positions are indexed.

CONSTRAINT

In spite of the fact that, for some systems, the distinguishing characteristic of a Bus from a Bundle is that it is indexed instead of having names for each position, the member Ports of a PortBus are also required to have names that are unique with respe ct to each other. This is because all PortBusses are PortBundles which have this constraint.

3.6.12 Entity: PortInst

```
TYPE

  cfidrPortInstOwner = SELECT(cfidrInst,cfidrPortInstBundle);

END_TYPE;
ENTITY cfidrPortInst
```

```
  ABSTRACT SUPERTYPE OF (ONEOF (
    (cfidrPortInstScalar, cfidrPortInstBundle))
  SUBTYPE OF (cfidrObject);

  Owner:        cfidrPortInstOwner;
  Describer:    cfidrPort;

DERIVE
  DescriberName:        cfidrStringT := Describer.Name;
END_ENTITY;
```

DESCRIPTION

The PortInst entity corresponds to a Port of a NetlistView which has been instantiated within another NetlistView. The Port which describes a PortInst is available through the *Describer* attribute. The *Name* and *Direction* attribute of t he describer Port are propagated to the *Name* and *Direction* attributes of the PortInst. A PortInst is owned by either an Inst or a PortInstBundle via the *Owner* attribute (through the cfidrPortInstOwner SELECT TYPE).

RATIONALE

A Net connected to a PortInst implicitly passes along its information to the describer Port and thus down one level in the design hierarchy. A PortInst is more like a pure reference than an Inst is with respect to the describer NetlistView.

PortInsts can have properties (via a set of Props) that are separate from any properties on the *Describer* Port. There is no automatic mechanism in the DR-PI whereby Props on a Port also appear on a PortInst. The Props of the describer Port may be obtained by getting the *Describer* Port then getting the *Props* set from it.

CONSTRAINT

An Inst has precisely one PortInst for each Port on the *Describer* NetlistView of the Inst. Each PortInst is of comparable sub-type to the corresponding Port: i.e., PortInstScalar for PortScalar, PortInstBundle for PortBundle, and PortInstBus for Po rtBus. Any PortInst substructure of bundles (or busses) must match the substructure of the *Describer* Ports.

3.6.13 Entity: PortInstScalar

```
ENTITY cfidrPortInstScalar
  SUBTYPE OF (cfidrPortInst);
  SELF\cfidrPortInst.Describer: cfidrPortScalar;

  NetScalar: OPTIONAL cfidrNetScalar;

DERIVE
  DescriberDirection:   cfidrPortDirection :=
                                        Describer.Direction;
END_ENTITY;
```

DESCRIPTION

As with PortScalar, the PortInstScalar entity models the lowest level PortInst. A PortInstScalar will have no further sub-parts. A PortInstScalar is directly connected to a NetScalar via its NetScalar attribute.

RATIONALE
The PortInstScalar is a reflection of the PortScalar only associated with Inst entities. The rationale is the same as for PortScalars.

CONSTRAINT
The *Describer* of the PortInstScalar must be a PortScalar.

3.6.14 Entity: PortInstBundle

```
ENTITY cfidrPortInstBundle
  SUBTYPE OF (cfidrPortInst);

 SELF\cfidrPortInst.Describer: cfidrPortBundle;
 PortInsts:  LIST [0:?] OF UNIQUE cfidrPortInst;
 NetBundles:  SET [0:?] OF cfidrNetBundle;
END_ENTITY;
```

DESCRIPTION
A PortInstBundle entity models the ability to group many PortInsts together and treat the group as a single entity under appropriate conditions. PortInstBundles have substructure. That is, a PortInstBundle contains other PortInst entities which are held i n the *PortInsts* attribute. A PortInstBundle may be made up of PortInstScalars or other PortInstBundles hierarchically. A PortInstBundle is connected to 0 or more NetBundles via its NetBundles attribute.

RATIONALE
PortInstBundles are a reflection of PortBundles only associated with the Inst entity. The rationale is the same as for PortBundles.

CONSTRAINT
The *Describer* of the PortInstBundle must be a PortBundle.

3.6.15 Entity: PortInstBus

```
ENTITY cfidrPortInstBus
  SUBTYPE OF (cfidrPortInstBundle);
DERIVE
  Start: cfidrInt32T := Describer.Start;;
  Step:  cfidrinst32T := Describer.Stop;
END_ENTITY;
```

DESCRIPTION
A PortInstBus entity is simply a PortInstBundle with additional attributes that associate a specific index value with each position in the bundle.

Start **is the index value for** ***Position*** **3.**
Step **is the difference in index values of** ***Position*** **n and n+1.**

RATIONALE
Many different terminologies were discussed in the DR TSC for non-scalar or composite signals: bundles, arrays, groups, and busses were the most common terms. In most cases, regardless of name, the Bundle entity, an ordered grouping of items (unique in the case of PortInsts), has the correct semantics while all other terminology distinctions are based largely on naming conventions.

The only semantic distinction on which agreement could be reached was that, in addition to the characteristic of being an ordered group, sometimes these groups also have an index to identify each position, that the index usually changes monotonically in v alue, and that the index of position 0 might be any value. The term Bus is used for these groups in which the positions are indexed.

CONSTRAINT
In spite of the fact that, for some systems, the distinguishing characteristic of a Bus from a Bundle is that it is indexed instead of having names for each position, the member PortInsts of a PortInstBus are also required to have names which are unique w ith respect to each other. This is because all PortInstBusses are PortInstBundles which have this constraint.

The ***Describer*** **of the PortInstBus must be a PortBus.**

3.6.16 Entity: Net

```
ENTITY cfidrNet

  SUBTYPE OF (cfidrNamedObject);

  ABSTRACT SUPERTYPE OF (ONEOF (

     (cfidrNetBundle, cfidrNetScalar));


  Owner:      cfidrNetlistView;

  Ports:      SET [0:?] OF cfidrPort;

  PortInsts:  SET [0:?] OF cfidrPortInst;

  NetBundles: SET [0:?] OF cfidrNetBundle;


WHERE

  SIZEOF(Ports) +  SIZEOF(PortInsts) >= 2;

END_ENTITY;
```

DESCRIPTION
The Net passes information throughout the design hierarchy by connecting Ports and PortInsts together. A Net is owned by a NetlistView via the ***Owner*** **attribute. A Net may be hierarchically contained by zero or more NetBundles available through the < I>NetBundles attribute.**

RATIONALE
The Net entity models the basic idea of connectivity. A net represents a set of connected things, namely, Ports and PortInsts. When two Ports and/or PortInsts are connected to a

Net, the implication is that those Ports and PortInsts will all "see" the sam e signal information in the context of the circuit being represented.

CONSTRAINT
Each net must connect at least two things---eventually.

3.6.17 Entity: NetScalar

```
ENTITY cfidrNetScalar
  SUBTYPE OF (cfidrNet);

  SELF\cfidrNet.Ports: SET [0:?] OF cfidrPortScalar;
  SELF\cfidrNet.PortInsts: SET [0:?] OF cfidrPortInstScalar;
  IsGlobal:      cfidrBooleanT;

END_ENTITY;
```

DESCRIPTION
The NetScalar entity models the lowest level Net which has no further sub-parts. A NetScalar connects to PortScalars and/or PortInstScalars. Multiple PortScalars connected to the same NetScalar short the PortScalars and will cause aliasing of NetScalars a t the next level up in the hierarchy whenever the *Owner* NetlistView is used as an Inst and the corresponding PortInsts are connected to Nets at that level.

A NetScalar is a "global net" when *IsGlobal* is TRUE. All global NetScalars with the same name are the same electrical net.

RATIONALE
To support the notion of Bundled signals, a concept is needed which represents a scalar signal value, but which has no further sub-signal components.

3.6.18 Entity: NetBundle

```
ENTITY cfidrNetBundle
  SUPERTYPE OF (cfidrNetBus)
  SUBTYPE OF (cfidrNet);

  SELF\cfidrNet.Ports: SET [0:?] OF cfidrPortBundle;
  SELF\cfidrNet.PortInsts SET [0:?] OF cfidrPortInstBundle;
  Nets: LIST [0:?] OF cfidrNet;
DERIVE
  Size: SIZE_OF(Nets);
END_ENTITY;
```

DESCRIPTION
The NetBundle entity models aggregation of Net entities. The member Nets of a NetBundle are available via the *Nets* attribute. NetBundles may contain NetBundles to model multiple

levels of Net hierarchy.

RATIONALE
The NetBundle is needed to allow hierarchical and other aggregate signal structures to be represented in a consistent manner. Bundles allow groups of signals to be treated as a single entity when it is appropriate and expedient to do so.

3.6.19 Entity: NetBus

```
ENTITY cfidrNetBus
  SUBTYPE OF (cfidrNetBundle);

  Start: cfidrInt32T;

  Step:  cfidrInt32T;
END_ENTITY;
```

DESCRIPTION
A NetBus entity is simply a NetBundle with additional attributes that associate a specific index value with each position in the bundle.

***Start* is the index value for *Position* 0.**
***Step* is the difference in index values of *Position* n and n+1.**

RATIONALE
Many different terminologies were discussed in the DR Working Group for non-scalar or composite signals: bundles, arrays, groups, and busses were the most common terms. In most cases, regardless of name, the Bundle entity, an ordered grouping of items (non-unique in the case of Nets), has the correct semantics while all other terminology distinctions are based mostly on naming conventions.

The only semantic distinction on which agreement could be reached was that, in addition to the characteristic of being an ordered group, sometimes these groups also have an index to identify each position, that the index usually changed monotonically in v alue, and that the index of position 0 might be any value. The term Bus is used for these groups in which the positions are indexed.

CONSTRAINT
In spite of the fact that, for some systems, the distinguishing characteristic of a Bus from a Bundle is that it is indexed instead of having names for each position, the member Nets of a NetBus are also required to have names which are unique with respec t to all Net names in the *Owner* NetlistView. This is because NetBusses are NetBundles which are Nets which have this constraint.

Table of Contents Next Chapter

Table of Contents Previous Chapter

4 Global Policy and Conventions

This section describes those topics that concern the overall functionality of the programming interface. It outlines the decisions (or *policies*) about which functions are available in the interface, how they behave in general, and what limitations there might be. When appropriate, rationale is given for the policy so that the Design Representation Working Group (DR WG) can use the information in future discussions.

4.1 Data Types

This section discusses the information that is included in the cfidr.h header file which (1) must be included in any code using the DR-PI and (2) in turn includes cfi.h with the CFI-wide type definitions. See "Appendix B Standard Header Files" for listings of the standard cfidr.h and cfi.h files.

4.1.1 Primitive Data Value Types

Boolean Values

```
typedef enum {
  CFIDR_FALSE  = CFI_FALSE,
  CFIDR_TRUE   = CFI_TRUE
} cfidrBooleanT;
```

NOTE: CFI_FALSE and CFI_TRUE are defined in cfi.h.

32-bit Integer Values

```
#define CFIDR_MAX_INT32        2147483647
#define CFIDR_MIN_INT32        -2147483647


typedef cfiLongT    cfidrInt32T;/* defined in cfi.h */
```

32-bit Floating Point Values

```
#define CFIDR_MIN_FLOAT32    -1E+37
#define CFIDR_MAX_FLOAT32    1E+37


typedef cfiFloatT    cfidrFloat32T;  /* defined in cfi.h */
```

Character Strings

```
typedef cfiStringT    cfidrStringT;  /* defined in cfi.h */
```

Void Return Values

```
typedef cfiVoidT    cfidrVoidT;  /* defined in cfi.h */
```

Object Identifiers

```
typedef cfiObjectIdT    cfidrObjectIdT;  /* defined in cfi.h */
typedef cfidrObjectIdT    cfidrNamedObjectIdT;
typedef cfidrObjectIdT    cfidrLibIdT;
typedef cfidrObjectIdT    cfidrCellIdT;
typedef cfidrObjectIdT    cfidrPortIdT;
typedef cfidrObjectIdT    cfidrPortBundleIdT;
typedef cfidrObjectIdT    cfidrPortBusIdT;
typedef cfidrObjectIdT    cfidrPortScalarIdT;
typedef cfidrObjectIdT    cfidrInstIdT;
typedef cfidrObjectIdT    cfidrPortInstIdT;
typedef cfidrObjectIdT    cfidrPortInstBundleIdT;
typedef cfidrObjectIdT    cfidrPortInstBusIdT;
typedef cfidrObjectIdT    cfidrPortInstScalarIdT;
typedef cfidrObjectIdT    cfidrNetIdT;
typedef cfidrObjectIdT    cfidrNetBundleIdT;
typedef cfidrObjectIdT    cfidrNetBusIdT;
typedef cfidrObjectIdT    cfidrNetScalarIdT;
typedef cfidrObjectIdT    cfidrPropIdT;
typedef cfidrObjectIdT    cfidrViewIdT;
typedef cfidrObjectIdT    cfidrEncapsulatedViewIdT;
typedef cfidrObjectIdT    cfidrNetlistViewIdT;
```

Iterator Identifiers

```
typedef cfiIteratorIdT     cfidrIterIdT;/* defined in cfi.h */
typedef cfidrIterIdT    cfidrCellsIdT;
typedef cfidrIterIdT    cfidrInstsIdT;
typedef cfidrIterIdT    cfidrLibsIdT;
typedef cfidrIterIdT    cfidrNetsIdT;
typedef cfidrIterIdT    cfidrNetBundlesIdT;
typedef cfidrIterIdT    cfidrPortsIdT;
typedef cfidrIterIdT    cfidrPortInstsIdT;
typedef cfidrIterIdT    cfidrPropsIdT;
typedef cfidrIterIdT    cfidrViewsIdT;
```

4.1.2 Enumerated Constants

Types of Objects

```
typedef enum {
 = 0,   CFIDR_UNDEFINED_OBJECTTYPE
 = 1,   CFIDR_LIB
 = 2,   CFIDR_CELL
 = 3,   CFIDR_PORTBUNDLE
 = 4,   CFIDR_PORTBUS
 = 5,   CFIDR_PORTSCALAR
 = 6,   CFIDR_INST
 = 7,   CFIDR_PORTINSTBUNDLE
 = 8,   CFIDR_PORTINSTBUS
 = 9,   CFIDR_PORTINSTSCALAR
 = 10,   CFIDR_NETBUNDLE
 = 11,   CFIDR_NETBUS
```

```
 = 12,    CFIDR_NETSCALAR
 = 13,    CFIDR_PROP
 = 14,    CFIDR_NETLISTVIEW
 = 15,    CFIDR_ENCAPSULATEDVIEW
 = 16     CFIDR_MAX_OBJECTTYPE
} cfidrObjectTypeT;
```

Types of Views

The string "CFIDR_NETLISTVIEW" is reserved as the value of the ViewType attribute for views of type cfidrNetlistViewIdT. The **cfidrObjectGetObjectType()** function can be used to distinguish between the NetlistView and EncapsulatedView types.

Types of Values

```
typedef enum {
  CFIDR_UNDEFINED_VALUETYPE  = CFI_UNDEFINED_VALUETYPE,
  CFIDR_INT32  = CFI_LONG,
  CFIDR_STRING  = CFI_STRING,
  CFIDR_FLOAT32  = CFI_FLOAT,
  CFIDR_BOOLEAN  = CFI_BOOLEAN,
  CFIDR_MAX_VALUETYPE
} cfidrValueTypeT;
```

NOTE: All the values for the right hand sides above starting with CFI_... are defined in cfi.h.

Types of Port Directions

```
typedef enum {
 = 0,    CFIDR_UNDEFINED_PORTDIRECTION
 = 1,    CFIDR_INPUT
 = 2,    CFIDR_OUTPUT
 = 3,    CFIDR_IO
 = 4     CFIDR_MAX_PORTDIRECTION
} cfidrPortDirectionT;
```

Types of Attach Modes

```
typedef enum {
 = 0,    CFIDR_UNDEFINED_ATTACHMODE
 = 1,    CFIDR_ATTACH_BY_ORDER
 = 2,    CFIDR_ATTACH_BY_NAME
 = 3     CFIDR_MAX_ATTACHMODE
} cfidrAttachModeT;
```

Types of Access Modes

```
typedef enum {
 = 0,    CFIDR_UNDEFINED_ACCESSMODE
 = 1,    CFIDR_READ
 = 2,    CFIDR_UPDATE
 = 3     CFIDR_MAX_ACCESSMODE
} cfidrAccessModeT;
```

Types of Iterator Modes

```
typedef enum {
 = 0,    CFIDR_UNDEFINED_ITERMODE
 = 1,    CFIDR_ITER_SCALARS
 = 2,    CFIDR_ITER_BUNDLES
 = 3,    CFIDR_ITER_TOP
 = 4,    CFIDR_ITER_ALL
 = 5     CFIDR_MAX_ITERMODE
} cfidrIterModeT;
```

Types of Errors and the Error Text

The following table lists possible error codes in numerical order along with the conditions under which this error is to be returned. Also shown is an example of the text that could be returned by the cfidrPIGetErrorText function. However, the exact wording of the text may be left to the implementor.

Table 1: **Error Codes**

Types of Errors		Example Error Text
typedef enum {		cfidrStringT cfidrErrorText[] = {
CFIDR_NO_ERROR	=0,	"No error occurred."
CFIDR_DESCRIBER_PORT_NOT_FOUND	=1,	"Describer Port associated with PortInst not found."
CFIDR_DESCRIBER_RECURSION	=2,	"Describer would recursively instantiate a view."
CFIDR_DESCRIBER_VIEW_NOT_FOUND	=3,	"Describer View associated with Inst not found."
CFIDR_INTERNAL_SYSTEM_ERROR	=4,	"Internal error occured in system implementing PI."
CFIDR_INVALID_ATTACH_MODE	=5,	"The attachment mode specified is out of the range of possible values."
CFIDR_UNUSABLE_DESCRIBER_OID	=6,	"The Describer OID refers to an unusable object"
CFIDR_INVALID_DESCRIBER_TYPE	=7,	"The Describer OID refers to an object that is not a view",
CFIDR_INVALID_DIRECTION	=8,	"The Direction is not a valid Port Direction."
CFIDR_INVALID_ERROR_CODE	=9,	"The Error Code constant specified is out of the range of possible values."
CFIDR_INVALID_VALUE	=10,	"The Value specified is out of the range of possible values."
CFIDR_INVALID_ITER	=11,	"The Iterator ID refers to a non-existent iterator."
CFIDR_INVALID_ITER_TYPE	=12,	"The Iterator ID refers to an iter ator over the wrong object type."
CFIDR_INVALID_MEMBERS	=13,	"Member is not a valid array of member names"
CFIDR_INVALID_NAME	=14,	"Name is not a valid string or contains illegal characters."
CFIDR_INVALID_OBJECTTYPE	=15,	"The type of object specified cannot be handled by the function."
CFIDR_UNUSABLE_OID	=16,	"The Object ID does not refer to a usable object."
CFIDR_UNUSABLE_PORT_OID	=17,	"The Port OID does not refer to a usable object."

```
CFIDR_INVALID_PORT_TYPE          =18,   "The Port OID refers to an object
                                        that is not a usable Port."
CFIDR_UNUSABLE_PORTINST_OID      =19,   "The PortInst OID does not refer
                                        to a usable object."
CFIDR_INVALID_PORTINST_TYPE      =20,   "The PortInst OID refers to an
                                        object that is not a usable
                                        PortInst."
CFIDR_INVALID_VIEWTYPE           =21,   "The ViewType specified is not a
                                        legal string or is the string
                                        reserved for the NetlistView."
CFIDR_LIB_ALREADY_OPEN           =22,   "The library specified is already
                                        open."
CFIDR_NEW_DESCRIBER_PORT_FOUND   =23,   "A Port exists on the Describer
                                        View that does not exist as a
                                        PortInst."
CFIDR_NO_NAME_MATCHES            =24,   "No matches were found
                                        between the Scalar names when
                                        trying to attach."
CFIDR_NAME_IN_USE                =25,   "An object with the same name
                                        already exists in this
                                        namespace."
CFIDR_OBJECT_ALREADY_EXISTS      =26,   "The object that would be
                                        created already exists in the
                                        applicable scope."
CFIDR_OBJECT_NOT_FOUND           =27,   "The desired object could not be
                                        found."
CFIDR_PORTINST_ALREADY_ATTACHED  =28,   "The PortInst is already attached
                                        to a Net - it cannot be connected
                                        again."
CFIDR_PORTINST_NOT_ATTACHED      =29,   "The PortInst specified must be
                                        attached in order to be
                                        detached."
CFIDR_PORT_ALREADY_ATTACHED      =30,   "The Port is already attached to a
                                        Net - it cannot be connected
                                        again."
CFIDR_PORT_MEMBERS_DIFFER        =31,   "The Describer Port members
                                        differ from associated PortInst
                                        members."
CFIDR_PORT_NOT_ATTACHED          =32,   "The Port specified must be
                                        attached in order to be
                                        detached."
CFIDR_VIEW_ALREADY_OPEN          =33,   "The View specified is already
                                        open."
CFIDR_OPEN_FOR_UPDATE_FAILED     =34,   "Opening the object in update
                                        mode failed."
CFIDR_INVALID_ITERMODE           =35,   "The IterMode specified is out of
                                        the range of legal values."
CFIDR_ILLEGAL_OPERATION          =36,   "The attempted operation is not a
                                        legal operation."
CFIDR_CROSS_VIEW_OPERATION       =37,   "Cannot connect objects not in
                                        the same view."
CFIDR_SCALAR_BUNDLE_ATTACHMENT   =38,   "Cannot attach bundles to
                                        scalars."
CFIDR_LIB_OPEN_FAILED            =39,   "The library could not be
                                        opened."
CFIDR_VIEW_OPEN_FAILED           =40,   "The View could not be
                                        opened."
CFIDR_INVALID_ACCESSMODE         =41,   "The Access Mode specified is
                                        out of the range of legal values."
CFIDR_READ_ONLY                  =42,   "The containing entity (library or
```

```
                                    View) has not been opened for
                                    update."
CFIDR_INVALID_KEY           =43,    "Key is not a valid string or
                                    contains illegal characters."
CFIDR_INVALID_NET_TYPE      =44,    "The OID is not of type net."
CFIDR_INVALID_PORTBUNDLE_TYPE =45,  "The OID is not of type
                                    portbundle."
CFIDR_INVALID_POSITION      =46,    "The position is not in the range
                                    0 to one less than the size of the
                                    bundle"
CFIDR_MEMBER_RECURSION      =47,    "The Bundle would contain itself
                                    or a sub-bundle of itself as a
                                    member."
CFIDR_MULTIPLE_VIEWS_MATCH  =48,    "There are multiple views of the
                                    same name in the same library or
                                    multiple views of the same type
                                    in the same library."
CFIDR_NO_VIEW_MATCHES       =49,    "No matches were found for
                                    views meeting the view selection
                                    criteria specified by the
                                    selectorSet."
CFIDR_UNUSABLE_NET_OID      =50,    "The Net OID does not refer to a
                                    usable object."
CFIDR_UNUSABLE_PORTBUNDLE_OID =51,  "The PortBundle OID does not
                                    refer to a usable object."
CFIDR_MAX_ERROR             =52     ""
} cfidrErrorT;                      };
-------------------------------------------------------------------------
```

4.2 Object Identifiers

The Design Representation Programming Interface functions refer to objects described by the information model (IM) by using object identifiers (or *OIDs*). An OID references a unique occurrence of an entity type defined in the IM. For example, each specific Library is acted upon using PI functions that are supplied with its OID. As another example, each Net owned by a particular View will have a unique OID.

4.2.1 OIDs Are Abstract Data Types

Policy: Users of the DR-PI must treat OIDs as an abstract data type. Nothing can be assumed about their implementation.

Rationale: The application cannot assume what the OID represents. In the long-term, the DR Working Group needs to understand if and how systems with differing OID lengths can be interoperable in a CFI framework.

NOTE: In CFI DR 1.0, OIDs are a fixed size for any given *binary instruction set machine architecture* and specified to be **twice** the size of an address (void *). However, **do not assume** that the OID is a structure containing two addresses even though that is the correct size for it.

4.2.2 OIDs Are Scoped Per Session

Policy: OIDs are valid for the current PI session. A session is defined as the interval of time which starts when the PI is initiated with a call to **cfidrPIInit()** and terminates when you dismiss the PI with a call to **cfidrPIQuit()**. Thus, OIDs are not persistent across multiple invocations of the PI. Also, the OID assigned to an object in one invocation cannot be assumed to be assigned to the same object in the next invocation.

4.2.3 OIDs Are Scoped by Open/Purge and Destroy

Policy: In addition to the lifetime of OIDs being limited to the PI session, the lifetime of OIDs is also limited by the open/purge operations (whose policies are described later) as well as by destroy operations. Thus, when an OID comes into existence via an open, it remains usable as a reference to the object until the object is purged or destroyed.

Rationale: The PI allows for creation, modification, and destruction as well as data retrieval. The DR Working Group also notes the need to incorporate the notion of opening, saving, and purging of data. The lifetime of OID usability must fit into these additional operations.

4.2.4 OIDs for Supertypes (Allow Polymorphism)

Policy: The Information Model defines a class hierarchy of entity types. The class hierarchy is defined by supertypes and subtypes. Some of the supertypes are ABSTRACT but the only objects which can actually be created are the non-abstract types, most of which have no further subtypes. The only exception is that each of the Bundle types (Net, Port, PortInst) can be created although each is also a supertype of a corresponding Bus, which can also be created. However, even though most supertypes can not be directly created, some PI functions are defined which accept OIDs of those supertypes in the IM. The important fact is that a supertype OID can refer to objects of *any* of the supertype's subtypes (or subtypes of those types).

For example, cfidrPortIdT is the OID of a supertype. Since the Port supertype specializes into PortBundle and PortScalar subtypes, a cfidrPortBundleIdT or cfidrPortScalarIdT can be passed to any function that takes as input a cfidrPortIdT. Furthermore, since a cfidrPortBusIdT is the OID of a PortBus which is the subtype of a PortBundle so it can also be passed to any of the functions that require cfidrPortBundleIdT for the type of an argument.

Likewise, a function declared to return a cfidrPortIdT can can also return a cfidrPortBundleIdT, cfidrPortBusIdT, or a cfidrPortScalarIdT. A call to **cfidrObjectGetObjectType()** will return the lowest subtype of object given any OID (i.e., it takes as input the **cfidrObjectIdT**).

Rationale: One way to reduce the number of functions generated by the PI is to define OIDs which may be of several different object types. Allowing OIDs to refer to supertypes is the natural way to do this. It allows a very object-oriented polymorphism in the PI functions.

4.2.5 Checking for the Same Object

Policy: There is no requirement that a given object only have one OID. Therefore a function is provided, **cfidrObjectIsSame()**, which determines whether two OIDs refer to the exact same object. This is the only mechanism that an application using the PI can use to determine this condition.

Rationale: Using an operator such as "==" is not valid since the OID is an abstract data type. Furthermore some implementations use more than one OID for the same actual object. This test for sameness must be performed by the implementation of the PI which knows what objects the OIDs are actually referencing.

4.2.6 Checking for a Usable Object ID

Policy: A function exists, **cfidrObjectIsUsable()**, which determines whether an OID is usable (i.e., refers to an existing object). This is the only mechanism that an application using the PI can use to determine this condition. An OID may become unusable if the object to which it is referring has been purged or destroyed.

Rationale: Since the application must treat the OID as an abstract data type, nothing can be inferred about the OIDs value and whether it refers to an existing object or not. This function allows testing for "dead" OIDs that no longer reference a usable object.

4.2.7 Checking for a Null Object ID

Policy: A function exists, **cfidrObjectIsNull()**, which determines whether an OID is the Null OID. A Null OID is not usable (i.e., it does not refer to an object), but it is valid as an OID. Null OIDs are returned to the application primarily when an error of some sort occurs during execution. This function is the only mechanism that an application using the PI can use to determine whether an OID is Null.

Rationale: Since an OID is an abstract data type, nothing can be inferred about what constitutes a Null OID.

Policy: There must be only one unique representation of a Null OID per session.

4.2.8 Iterator IDs Are Not OIDs

Policy: Many attributes in the information model represent one-to-many relationships. Functions in the PI returning objects that participate in the "many" side of the relationship are done via returning a Iterator ID. Iterator IDs are not OIDs in the sense that they do not refer to actual design objects in the database. By using functions beginning with "cfidrIterNext...," the objects traversed by the Iterator are returned one at a time. Also, each time a one-to-many relationship is traversed, a unique Iterator is created. Thus, several independent traversals can be made over the same relationship, each traversal uniquely identified by its own Iterator ID.

NOTE: The Iterator "objects" are used to traverse both SETs and LISTs as specified by *EXPRESS*. SETs have no significant order while the order of LISTs is significant.

4.3 Error Handling

4.3.1 Error Handling Supports Functional Programming Style

Policy: Error handling in the DR-PI supports a functional programming style instead of a procedural programming style. Thus, the normal return value of functions is the data being sought, e.g., an OID, an integer, a string, etc. The return value often does not indicate that an error has occurred. In some cases that is because there is no illegal value in the range of possible return values. Hence the user should always check the error return parameter to determine if an error has occurred.

4.3.2 Error Code Output by Each Function

Policy: A pre-defined error code type is defined for the DR-PI. A pointer to a variable of this type is required as the last argument of each function in the PI. The argument is passed-by-reference (i.e., the address of the variable is supplied) and is output from each function. Each function description enumerates a list of possible error codes returned.

Rationale: The Specification Standards Working Group (SSWG) recommended defining the error code as an output of each function so that the functional interface style can be accommodated. Also, by forcing the error code to be output as a specific argument with each function, it will hopefully provide incentive for applications to check the error code. This checking was commonly bypassed when a global error code was used.

4.3.3 Pre-defined Error Code Text Available

Policy: Each error code will have pre-defined text associated with it. The application may wish to enhance this error text with information specific to the context in which the error occurred.

Rationale: Pre-defining the error text relieves the application of creating error messages when the pre-defined text is sufficient.

4.4 Names and Strings

4.4.1 Characters in Names Have No Semantics

Policy: There are NO SEMANTICS associated with any names or the characters used in them. Thus if a NetBus has a *Name* attribute "ADDR[31:0]" it may NOT be assumed that the *Start* attribute is 31 or that the derived *Width* attribute is 32 or that the *Step* attribute is -1. Also, an implementation should not interpret "/", "\", or ".." (or any other characters) as file system directory name separators.

4.4.2 Character Set for Name Strings is Restricted

Policy: The valid character set for objects having a *Name* attribute is:

- The Printable ASCII character set.

```
sp   !   "   #   $   %   &   '
(   )   *   +   ,   -   .   /
0   1   2   3   4   5   6   7
```

```
8   9   :   ;   <   =   >   ?
@   A   B   C   D   E   F   G
H   I   J   K   L   M   N   O
P   Q   R   S   T   U   V   W
X   Y   Z   [   \   ]   ^   _
`   a   b   c   d   e   f   g
h   i   j   k   l   m   n   o
p   q   r   s   t   u   v   w
x   y   z   {   |   }   ~
```

No other characters are allowed to be part of the *Name* attribute. Any function which takes a *Name* attribute as input for creation or modification must check to ensure this policy is upheld.

Note that Null strings (" ") are legal input parameters to certain ViewSelection functions.

4.4.3 Character Set for Other Strings Is Unlimited

Policy: For any other object attribute which is a string, there is no limitation on which characters comprise the string other than that the string be delimited with the Null character (i.e., '\0').

Rationale: Since a string is just a character array and each character is at least an 8-bit structure, the application may store any bit sequence in the memory which constitutes the character. Thus, the character array may be comprised of any binary sequence, as long as it is delimited by the Null character.

4.4.4 String Comparison Is Case-Sensitive

Policy: For all strings (especially those referring to *Name* attributes), the application must treat comparison of the strings as case-sensitive.

Rationale: Since case-sensitive strings can be added, it is consistent to expect their comparison to also be case-sensitive.

NOTE: It is strongly recommended that unique names differ in other than just case.

4.4.5 A String's Maximum Length Is Not Defined

Policy: There is no defined limitation for the length of a string.

Rationale: There was no one maximum length on which the Working Group participants could agree. Some implementations of the PI will be unable to support unlimited lengths. An implementation will return an error code denoting a system-dependent failure if a string exceeds the capacity of that system and this is detectable by the implementation.

4.4.6 Strings Supplied to Functions Are "Stored"

Policy: The memory pointed to by a string (i.e., char *) that is passed into a PI function must remain

valid until the function has exited. After the function has exited it may then be changed without affecting the data "stored" by the PI.

For example, the function **cfidrPropSetString()** value must be supplied a pointer to a string as the *value* argument. The memory pointed to by this argument should not be changed until after the function has exited. However, this memory may then be changed without affecting the "stored" value for the string. Specifically, if the cfidrPropSetString() has exited, then the memory that was pointed to is changed, and then **cfidrPropGetString()** is called (on the same Prop object), the value returned by cfidrPropGetString () will be the same as that supplied to the last call to cfidrPropSetString().

Rationale: This prevents the user of the DR-PI from having to create separate storage for every unique string ever created. It presumes the provider of the DR-PI has a place to store these values.

4.4.7 Strings Returned May All Be in One Buffer

Policy: The memory pointed to by a string that is returned by a PI function is only guaranteed to remain valid until the execution of another PI function which also returns a string or until the function **cfidrPIQuit()** is called.

For example, if **cfidrNamedObjectGetName()** returns a string representing the *Name* attribute of an object, that string remains valid until the next call to a DR-PI function returning a string, which might be **cfidrNamedObjectGetName()**, or **cfidrPropGetStringValue()** (gets a property's string value), or any other function returning a string. Calling **cfidrPropGetBooleanValue()** would not invalidate the *Name* since that function returns a Boolean.

Rationale: This allows the provider of the DR-PI to use one string buffer for the return of all strings. It requires the user of the DR-PI to make a copy of any string that will still be needed before making a call to another string-returning DR-PI function.

However, while it is not required that a PI provider use more than one buffer in which to return strings, it is recommended that at least several (say 10) of the most recently returned strings remain valid in spite of another string-returning call. This allows for the possibility of a user failing to copy a needed string before calling a second string-returning PI function.

4.5 Persistency Behavior (Open / Save / Purge)

4.5.1 Open/Save/Purge Operate Only on Libraries and Views

Policy: The operations of open, save, and purge are applicable only to Library, View, and SelectorSet objects. Thus, the functions implementing these operations should only accept input that defines Libraries, Views, or SelectorSets.

Rationale: Since the PI has to deal with creation, modification, and destruction, that means it is also able to persistently deal with object data. Also, the DR Working Group notes that open, save, and purge are generic operations that apply to any information model.

An arbitrary decision was made to define which objects these operations acted on. The decision was

based on Views being the smallest unit that may be opened. Libraries were the highest level. Cells were not included because of the behavior of open (see below). SelectorSets are equivalent to Libraries with respect to persistence behavior of View Selection objects. They have no owner but own all of the Selectors "below" them.

It is expected that in the future, these operations will be replaced by operations provided by a generic CFI standard for data management.

4.5.2 Description of Open Behavior

Open Makes Objects Available

Policy: The open operation *makes available* the desired object.

Rationale: The term "makes available" is deliberately used so as not to put any requirement on whether the object is made available by bringing it into memory at the time of open or by leaving it on disk until the object is referenced.

Open Recursively Walks Ownership Hierarchy

Policy: In addition to making available the desired object, **open** makes available all the objects in the ownership hierarchy until another object is reached that must be opened individually (i.e., has its own **open** function).

Consider the Information Model and take into account the policy that only Libraries and Views can be opened. Opening a Library not only makes the Library object available, but also all of its owned Properties, the Cells owned by the Library, and the Cells' owned Properties. The Views owned by each Cell are not made available since Views must be opened by themselves.

Opening a View makes the View object available as well as all its owned Properties, Nets, Insts, Ports, etc.

Rationale: The policy was derived arbitrarily but a consistent and well-defined behavior is presented.

Open Modes

Policy: There are two defined modes for open: CFIDR_READ and CFIDR_UPDATE. CFIDR_READ allows for read-only access to a Library or a View. No Cells, Views, or Properties on Libs or Cells, can be created or deleted from a Library unless the Library is opened for CFIDR_UPDATE. An existing View may be opened for CFIDR_UPDATE even though it's owning Library is opened for CFIDR_READ. A Lib can be destroyed regardless of the access mode of any object.

SelectorSets do not have an access mode, and they or their contents can be changed, purged, or destroyed regardless of the access mode of any object.

Until concurrence specifications are provided, the specification for open explicitly does not include support for multiple writers or multiple readers with a single writer.

Automatically Opening Libraries

Policy: When attempting to open a View, if the owning Library is not opened already, an attempt to open it should be made.

Rationale: This appears to be the appropriate thing to do instead of returning an error.

Policy: When traversing a set of Libraries using the **cfidrIterNextLib()** function, if the Library is not opened already, an attempt to open it should be made.

Automatically Opening Views

Policy: Views are automatically opened in two different situations: (1) when traversing the list of Views owned by a Cell, and (2) when traversing the *Describer* relationship from an Inst to the View.

Rationale: To reduce the number of functions, the automatic feature of opening views was defined as part of the traversal functions instead of defining separate functions for traversing just the Views opened as opposed to all Views.

No Automatic Creation of Libraries and Views via Open

Policy: If an attempt to open a non-existent Library or View occurs, the **open** function returns an error. The Library or View is not created automatically.

Rationale: If the creation were automatic, the only way to determine whether the Library or View exists already is to iterate through the list of Libraries or Views looking for the one in question. This policy makes the application create the Library or View explicitly if it does not exist.

4.5.3 Description of Save Behavior

Save Makes Changes Persistent

Policy: The save operation synchronizes the persistent version of data on disk with any changes made to that data since the last save or open. This behavior pertains to destruction as well as creation and modification; however, destruction of the persistent copies of libs, views, or selector sets occurs regardless of any save. All OIDs of objects saved remain usable and unchanged after the save.

Save Walks Ownership Hierarchy

Policy: Similar to **open**, **save** also recursively walks the ownership hierarchy of objects rooted by the specified object and saves all the objects in this hierarchy. Unlike **open**, **save** continues walking the ownership hierarchy when it encounters an object which can be saved individually.

Consider the Information Model and take into account the policy that only Libraries and Views can be saved. Saving a View updates the persistent version of the View and its Nets, Ports, Insts, and Props. Saving a Library not only saves the Library and Cells, it also saves all the Views as well.

Rationale: When opening a Library, opening all the Views automatically is probably undesirable in most situations. However, when saving a Library, saving all the Views and thus taking a "snapshot" of the Library at that instant is most desirable. If necessary, the application can save each View individually.

4.5.4 Description of Purge Behavior

Purge Removes Changes

Policy: The purge operation removes any changes made to an object since the last save or open. The object then becomes unavailable and its OID becomes unusable. Opening the object again assigns the object an OID and makes it available for updating.

An example implementation that would provide this behavior would make an in-memory copy of an object. All changes would be made against the in-memory copy of the object. If that object is then "purged," the copy of the object is deleted from memory, leaving the original, on-disk copy unchanged.

A lib, view, or selector set may be purged at any time, unrelated to access mode.

Purge Walks Ownership Hierarchy

Policy: Similar to **open, purge** also recursively walks the ownership hierarchy of objects rooted by the specified object and purges all the objects in this hierarchy. Unlike **open, purge** continues walking the ownership hierarchy when it encounters an object which can be purged individually. Thus, purge behaves similarly to save in this respect.

Consider the information model and take into account the policy that only Libraries and Views can be purged. Purging a View removes the View's objects (Nets, Ports, Insts, and Props). Purging a Library not only purges the Library and Cells, it also purges all the Views as well.

Rationale: When opening a Library, opening all the Views automatically is probably undesirable in most situations. However, when purging a Library, it does not make sense to only purge the Library and Cells and not the Views.

4.5.5 No Close or Revert Operations

Policy: No functions to **close** or **revert** Libraries and Views are specified for the DR-PI.

Rationale: Although the DR Working Group discussed **close** and **revert** operations, no agreement was reached as to their detailed behavior---especially with **close**. If **revert** means to refresh the current objects available with their persistent definition, this can be modeled using **purge** followed by **open**.

4.6 Library Functionality

Multiple Libraries May Be Open at the Same Time

Policy: Multiple libraries may be open at the same time, thus allowing NetlistViews to have instances of NetlistViews contained in other libraries than themselves.

Libraries Retained Across Session Boundaries

Policy: After creating and saving a Library, the DR-PI retains the Library across session boundaries. Thus, the Library may be subsequently opened and updated in a future session.

Rationale: Hopefully this is obvious since persistency of the data is a primary goal of any database. OIDs, however, do not persist across sessions.

No Knowledge of Storage Hierarchy

Policy: The application using the PI requires no knowledge of the storage hierarchy with respect to where and how Libraries are contained in this hierarchy. The application sees Libraries only by the names given to the Library at creation.

Rationale: Each implementation of the PI probably has its own internal mapping of a Library onto a storage hierarchy. This system-dependent behavior should be hidden completely from the DR-PI user.

4.7 Effects of the Ownership Hierarchy

The ownership hierarchy plays an important part in the IM and is used by many of the PI functions. The creation functions define what object owns another. The **cfidrCellGetOwner(), cfidrInstGetOwner(), cfidrNetGetOwner(), cfidrPortGetOwner(), cfidrPortInstGetOwner(), cfidrPropGetOwner(),** and **cfidrViewGetOwner()** functions return an object's owner. The traversal of an object to all of the objects it owns constitutes one level of the ownership hierarchy. By starting at a Lib and recursively traversing to all owned objects, all design objects "in" that library will be visited at least once.

4.7.1 Open / Save / Purge Uses Ownership Hierarchy

As stated previously, the functions dealing with persistency (open, save, and purge) each work as though traversing down some or all of the ownership hierarchy. See the discussions of these policies for more details.

4.7.2 Destroy Uses Ownership Hierarchy

Policy: Similar to open, **destroy** also recursively walks the ownership hierarchy of objects rooted by the specified object and destroys all the objects in this hierarchy. Unlike open, **destroy** continues walking the ownership hierarchy when it encounters an object which can be destroyed individually. Thus, destroy behaves similarly to save and purge in this respect.

NOTE: The term **destroy** is used instead of **delete** to emphasize that the object is removed entirely and not just disconnected from all relationships in which it participates.

Rationale: It would not make sense, for example, when destroying a View, that the owned Nets, Insts, etc., are left. An object cannot exist without an owner.

4.8 "Describes" / "WhereUsed" Is Not Required

Policy: The inverse of the *Describer* attribute for Insts, sometimes called *Describes* or *WhereUsed* is neither specified in the IM nor available via a DR-PI function.

Rationale: This relationship cannot be fully maintained in general.

For example, consider a NetlistView "A" with an Inst "b1" of NetlistView "B". Suppose this is done, then "A" is stored, "B" is stored elsewhere, and then the storage media on which "A" resides is physically destroyed (or just taken off-line). An attempt of "B" to keep "b1" on a *Describes* attribute will result in a dangling reference because "b1" no longer exists (in the system).

For another example, consider a widely used "primitive" such as a basic Inverter. The *WhereUsed* list for this item could become extremely long---easily in the millions! If not impossible, it is generally unreasonable to require an implementation to keep track of all these uses.

NOTE: this neither precludes nor forbids a particular implementation from maintaining *Describes* or *WhereUsed* information for as broad of context as reasonable (usually all designs open in a session).

4.9 "Describer" / "Describes" Modification Behavior

4.9.1 Modification on "Described" Objects Disallowed

Policy: The DR-PI disallows any modifications to Described objects (i.e., Insts and PortInst) with respect to data derived from the *Describers* (Views and Ports respectively).

Thus, an Inst's *Name* attribute may change since it is defined only for the Inst. However, the Inst's *Describer* relationship cannot be updated. Likewise, a PortInst's *DescriberName* and *DescriberDirection* attributes cannot be modified since they refer to information derived from the *Describer* Port.

Rationale: Restricting the modifications simplifies issues of consistency between the Describer object and what it Describes. These limitations will be revisited in future versions of the DR-PI.

4.9.2 Limited Updates on "Describer" Allowed

"Describes" Object Not Affected by "Describer" Creation

Policy: *Describer* objects can be created without any affect on *Describes* objects that might otherwise know about the new *Describer* objects.

For example, creating another Port in the *Ports* of a NetlistView does not automatically add the appropriate PortInst to the *PortInsts* of any Inst whose *Describer* is this NetlistView.

Rationale: Limiting creation of *Describer* objects until they are first instantiated is impractical without full support of the *Describes* relationship. That some of the described Views may not even be opened (or openable!) complicates this issue even further.

Destroying "Describer" Invalidates "Describer" Object Relationship

Policy: When a *Describer* object is destroyed, the *Describer* relationship referencing it becomes invalid.

For example, destroying a NetlistView means that any Inst referencing that NetlistView via the *Describer* relationship must return a Null OID when traversing this relationship. The actual time at which the relationship is broken depends on the implementation. It may happen at the time of destruction, when opening the NetlistView owning the Inst, or when attempting the traversal.

Rationale: The only way to avoid defining this behavior is to disallow destruction. Allowing destruction of the *Describer* objects at the minimum means that trying to reference the *Describer* objects should fail.

Updating "Describer" Attributes Not Allowed

Policy: Analogous to not allowing modification of *Describes* objects, the PI disallows any updates to *Describer* object attributes.

Thus, for example, once a NetlistView is created, its *Name* may not change. The same is true for Port attributes.

Rationale: This limitation helps reduce the number of issues surrounding consistency between *Describer* and *Describes* objects.

However, describer attributes (including names of ports, their directions, start/step values, even to what nets they are connected) might implicitly be changed in server implementations that allow describer relationships to be re-established automatically "by-name" when a describer is destroyed and then re-created. (See "Re-Establishing Describer Relationships" below).

Re-Establishing Describer Relationships

There are currently no requirements or limitations with regard to re- establishing describer relationships with Insts after their describers have been destroyed. A server (CFI DR provider) is free to consider new view/port oids, created subsequent to the destruction of describer views/ports, as "replacement" describers. Naturally, since recreation of describers might result in differences in the number, names, attributes, attachments, or properties of objects in the view, a server must make clear the details of functionality in all possible cases that might occur.

Alternatively, a server is under no obligation to re-establish such "by-name" relationships, and may even forbid it completely, putting the burden of destroying and recreating all affected Insts solely on client applications.

Each server will document its own rules for describer relationship behavior. A client application may need to be aware of the differences among server implementations that could affect its operation and interoperability with other server implementations.

Functions to Check Consistency

Policy: To make it easier to determine whether an Inst or PortInst is out-of-date regarding its *Describer* View or Port, specific functions exist in the DR-PI: **cfidrInstCheckDescriber()** and **cfidrPortInstCheckDescriber()**.

Rationale: Rather than putting the burden on the application to determine this fact, a specific PI function

helps isolate the checking and make the determination more uniform across applications and PI implementations.

4.10 Limitations During One-to-Many Relationship Traversal

4.10.1 No Add/Destroy During List or Set Iteration

Policy: The application should not add or destroy any objects that participate in a List or Set when actively iterating over the objects in that relationship (via functions such as **cfidrIterNextCell()**, **cfidrIterNextNet()**, etc.). Adding or destroying objects has an undefined effect on whether the object shows up (or does not) during iteration.

Rationale: The DR Working Group has not resolved the details of adding/destroying with respect to active iterators other than determining that these operations cannot be done at all during active iteration.

4.11 Limitations on Top-Down Designing

4.11.1 Top-down Creation Is Not Directly Supported

Policy: When creating an Inst, the *Describer* NetlistView must already exist. PortInsts on the Inst are only created corresponding to the Ports on the *Describer* NetlistView. Thus, the DR-PI discourages and provides no direct support for top-down designing.

Rationale: The details of doing top-down design have not been resolved including what propagation of changes from Ports to PortInsts is necessary or desirable.

NOTE: It is possible to fake the appearance of top-down design by hiding the creation of a NetlistView for use as the Inst's *Describer* during the *apparently* (to the user) *direct* creation of the Inst. Then, to modify the PortInsts of the Inst, Ports are modified on the hidden *Describer* NetlistView and the Inst is deleted and re-created. Since this requires the tool to remember any Net to PortInst attachments, delete, and then re-create them, it is not an *ideal* way to do top-down design, but it is possible.

4.11.2 Creating an Inst Creates PortInsts

Policy: When creating an Inst, all of the Ports in the Describer View are automatically created and made available as PortInsts owned by the Inst. This is the *only* way that PortInsts are created.

Rationale: Rather than forcing the application to create each PortInst directly, the DR-PI does this work automatically. Also, the DR-PI wants to enforce as much consistency as possible between the Inst and its Describer View.

4.12 Limitations on Searching Functions

4.12.1 Search / Query Functions Are Only Defined For Names

Policy: Other than a simple set of **cfidrFind...ByName()** functions, the DR-PI does not define functions to search and find an object given one of the object's attributes.

Rationale: This functionality can be completely implemented using the one-many relationship traversal functions (e.g., **cfidrNetBundleGetNets()**, **cfidrViewGetNets()**, and **cfidrIterNextNet()**). After iterating over an object, the application can do any necessary comparison checking. Thus, the DR-PI already supports the primitive functionality necessary.

cfidrFind...ByName() functions have been added because of the potential for speedup they provide, especially with network access to the DR-PI via Remote Procedure Calls. Furthermore, since names are required to be unique in each name scope, any implementor must already have internal functions to check for the pre-existence of a potential new name. Thus, providing this function as part of the defined CFI DR-PI interface is not difficult in general. However, these functions have been kept deliberately simple. Therefore, there are no functions defined at this time for case-insensitive name lookup, or for wild cards, or for regular expressions.

4.13 Limitations on Object Creation

4.13.1 Owner Always Specified

Policy: Whenever creating an object, the PI creation functions require specification of the object's *Owner*. Thus, the PI disallows creating an object and assigning it an owner at some later time.

Rationale: Most systems currently cannot handle creating an object without an owner. To do so usually requires this capability be modeled in the system in an unclean manner. For those systems allowing object creation without an owner, the PI limitation of specifying the owner at creation time just combines the creation and owner assignment into one step instead of two, separate disjoint steps. This behavior does not require any form of workaround.

4.13.2 Inst/PortInsts Do NOT Automatically "Inherit" Props

Policy: Properties existing on the *Describer* NetlistView and its Ports are NOT inherited or automatically created on an Inst and its PortInsts respectively during creation of that Inst. Specifically this means that when getting the Props of the Inst or PortInsts, only those Props *explicitly* created on the Inst or PortInst via the DR-PI are required to appear.

Rationale: Not all systems automatically copy or can reference properties from the *Describer* when accessing the "Describee" Inst or PortInst. For those that can, the Inst appears to have both the properties directly attached to it and those of the *Describer* NetlistView. The same is done for each PortInst with respect to its *Describer* Port. The usual semantic is that, for a given name, that Prop on the Inst (or PortInst) "overrides" any on the *Describer*. If this behavior is desired then, when looking for a Inst (or PortInst) Prop, an application using the DR-PI should first get the directly attached Props. If the desired Property is not found, then the *Describer*'s Properties should be gotten.

4.14 Bundle/Scalar Manipulation

4.14.1 Name Change Not Allowed for Attached or Bundled Scalars

Policy: The DR-PI does not allow changing the *Name* attributes of Net or Port Scalars when they are contained in a bundle or already attached to a Port or Net.

Rationale: Name changes are not allowed since some implementations rely on the names to establish connectivity (and other relationships). In such systems changing the name can have side effects including changes in the circuit connectivity.

As with most of the other bundle- and scalar-related limitations, this helps provide a consistent and simple approach for specifying the PI functionality. There is insufficient agreement in the DR Working Group on the fine grain behavior associated with allowing maximum editability, especially with respect to connectivity. Restrictions present now may be lifted at a later time when this behavior can be standardized.

The DR-PI does now allow **attach** and **detach** connectivity operations on bundle members. In addition, changing bundle membership is allowed.

4.14.2 Attribute Modification on Unconnected Bundle Only

Policy: A bundle must be completely unconnected (i.e., all of the members unconnected) in order for any of the bundle's attributes to be modified.

This includes modification of a Bundle's *Name* attribute via **cfidrNamedObjectSetName()**. It also includes modification of the contents of the Bundle via **cfidr...BundleCreate...Bundle/Bus()**, **cfidrNetBundleInsert/RemoveNet()**, and **cfidrPortSetOwner...()**

Rationale: This condition simplifies specifying PI functionality regarding connectivity in the presence of incremental changes.

4.14.3 NetBundle Connection to Port[Inst]Bundle is a Derived Relationship

Policy: Only scalar connections (attachments) are explicitly made between the members of a NetBundle and a PortBundle or PortInstBundle. Bundles are only ever connected when at least one member is connected and not simply because a function was executed to connect them.

The functions that try to connect bundles by name or by position are really "power" operations that cause recursive traversal "down" the bundles looking for at least one scalar to connect. Then, if and only if a pair of scalars (one Net and one Port or PortInst) are finally connected, do the bundles in which the scalars reside get connected. Because of this rule, a function call to attach a NetBundle to a Port[Inst] Bundle where these bundles are actually contained in higher level bundles can cause also attachments to recursively happen "above" the bundles being explicitly connected.

The functions to disconnect bundles are also "power" operations looking for scalar connections with the

results of the scalar disconnections propogated "up" to any containing bundles possible causing bundle disconnections.

Functions to destroy bundle members or change their owner (**container**, in the case of NetBundles) are also followed by a recursive analysis both "up" and "down" the bundle hierarchy to determine the resulting bundle connectivity.

Table of Contents Next Chapter

Table of Contents Previous Chapter

5 Function Definitions

5.1 Document Conventions

The conventions used in this chapter are explained in the following sections. This includes the ordering of the function definitions, how function names are related to the information model, the font and style conventions, and the particular subsections used in the definition of each function.

5.1.1 Order of Presentation

The functions in this chapter are grouped by object type with the sections in alphabetical order by object name. However the object types Net, Port, and PortInst are each kept in a section containing not only that supertype but all of its subtypes. Thus Net, NetBundle, NetBus, and NetScalar functions are all together in one section: Net Functions. Similarly, Port, PortBundle, PortBus, and PortScalar are in the section: Port Functions. PortInst, PortInstBundle, PortInstBus, and PortInstScalar are also all in one section: PortInst Functions.

Within each object type section, the order of the functions is strictly alphabetical.

Note that two "pseudo-objects" are used in the DR-PI:

1. Functions which are addressed to the top-level environment instead of to a particular DR object are prefixed **PI**.
   Iterator is not a defined object in the Information Model, but it is the mechanism used to get the elements of Sets and Lists. All the Iterator operations use the object prefix **Iter**.

5.1.2 Naming and Style Conventions

Functions are named according to the Design Representation Working Group's understanding of the Specification Standards Working Group recommendations. Names are constructed from four parts:

Example 1 Example 2

1. prefix: cfidr cfidr
   target/subject: Cell NetBundle

This is always the object type of the first parameter, except for the functions whose target/subject is PI which is a virtual target and not supplied as an argument.

verb: Create Insert
return/object: View Net
In the case of **cfidrCellCreateView()**, View is the type of the object that is returned. In the case of **cfidrNetBundleInsertNet()**, the return type is void but the parameter *owner* is the NetBundle that, after

the function call, has the newly inserted Net (supplied as the *net* parameter) included in it.

Compound names for any of the above items have their sub-parts capitalized: e.g., NetBundle.

Entity instance references are capitalized: View.

Entity type IDs as declared are bold: **cfidrNetlistViewIdT**.

Relationship names are shown capitalized in italics: *Describer*

Formal parameter names are italicized lower case: *owner*.

Function name references are in bold followed by (): **cfidrNetlistViewGetPorts()**

Items whose PI names are abbreviated may be spelled out fully in textual descriptions: e.g, library for Lib, instance for Inst and iterator for Iter.

5.1.3 Section Names and Meaning

Each function description has up to nine sections:

DECLARATION

The ANSI C form of the function declaration.

DESCRIPTION

A text description of the operation performed by the function.

[2] NOTE: When a DR-PI function can be implemented entirely using other DR-PI functions, it is called a "Level 2" function. This is shown by [2] after the function name in the section title and by a note labeled like this.

RATIONALE

A text description of additional, non-obvious, items that pertain to the operation of this function.

RETURN VALUE

A text description of return value(s) for this function including the value to be returned in case of error conditions.

PARAMETERS

For each formal parameter, a description of that parameter including whether it is an input to or an output from the function.

ERROR CODES

In the case of the *error* parameter, which is the last parameter to every function, each of the possible error codes is shown along with the conditions under which this error is to be returned.

No precedence is implied by the order of error codes listed for any function. In the case where more than one error occurs, any of the error codes that apply may be returned.

PRE-CONDITIONS

Things that must be true before the function is executed.

POST-CONDITIONS

Additional things (beyond the main operation of the function) that are true after the function is executed.

A compliant implementation of the DR-PI will ensure that all of the behavior documented in the description is followed, including the pre- and post-conditions.

REFERENCES

This is all other DR-PI functions relevant to the operation of this function: those with partial or similar functionality or those which could be called by or could call this function.

5.2 Cell Functions

5.2.1 cfidrCellCreateEncapsulatedView

DECLARATION

```
cfidrEncapsulatedViewIdT cfidrCellCreateEncapsulatedView(
cfidrCellIdT owner,
cfidrStringT name,
cfidrStringT viewType,
cfidrStringT key,
cfidrErrorT *error)
```

DESCRIPTION

This function creates an EncapsulatedView owned by the Cell specified via *owner*. The EncapsulatedView has its *Name* attribute set to *name*, its *ViewType* attribute set to *viewType*, and its *Key* attribute set to *key*.

RETURN VALUE

The return value is a cfidrEncapsulatedViewIdT referring to the EncapsulatedView just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the Cell which owns the created EncapsulatedView.

name (input) The string representing the name of the EncapsulatedView.

viewType (input) The string representing a type classification of the EncapsulatedView. Example values are "VHDL behavior" or "Spice 14G Model".

key (input) The string representing a "key" for the EncapsulatedView. This is an implementation-specific token for accessing the EncapsulatedView. It may be a file name, a design data manager name, a persistent object identifier, or other such item.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a Cell.

CFIDR_READ_ONLY:
The library that owns owner is not open for update.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_INVALID_VIEWTYPE:
the viewType parameter is not a valid string or is the string reserved for the NetlistView: "CFIDR_NETLISTVIEW."

CFIDR_INVALID_KEY:
the key parameter is not a valid string.

CFIDR_OBJECT_ALREADY_EXISTS:
a View with the same name already exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

POST-CONDITIONS

Insts (in NetlistViews) are never created with EncapsulatedViews as their *Describer*.

A created EncapsulatedView is not persistent across session boundaries until it is saved.

REFERENCE

cfidrNetlistViewCreateInst()

5.2.2 cfidrCellCreateNetlistView

DECLARATION

```
cfidrNetlistViewIdT cfidrCellCreateNetlistView(
cfidrCellIdT owner,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function creates a NetlistView owned by the Cell specified via *owner*. The NetlistView has its *Name* attribute set to *name*.

RETURN VALUE

The return value is a cfidrNetlistViewIdT referring to the NetlistView just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the Cell which owns the created NetlistView.

name (input) The string representing the name of the NetlistView.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a Cell.

CFIDR_READ_ONLY:
The library that owns owner is not open for update.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_ALREADY_EXISTS:
a NetlistView with the same name already exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

POST-CONDITIONS

The ViewType attribute is automatically set to "CFIDR_NETLISTVIEW."

The **cfidrInstCheckDescriber()** function can be used to determine whether there are discrepancies between an Inst and its *Describer* NetlistView.

A created NetlistView is not persistent across session boundaries until it is saved.

REFERENCE

cfidrObjectDestroy()
cfidrNetlistViewCreateInst()
cfidrInstCheckDescriber()

5.2.3 cfidrCellFindViewByName

DECLARATION

```
cfidrViewIdT cfidrCellFindViewByName(
cfidrCellIdT cell,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the View with the specified name that is within the Views relationship defined for the Cell specified by cell.

RATIONALE

cfidrCellFindViewByName() as currently defined will search all Views, Encapsulated and Netlist. This is reasonable since the current Information Model says names are unique across all Views. But if the Information model is changed so that separate namespaces are defined for Encapsulated and Netlist Views, then two different functions will be needed.

RETURN VALUE

The return value is a cfidrViewIdT referring to the View just found. If an error occurs, a Null OID is returned.

PARAMETERS

cell (input) The OID of the Cell from which the View is to be found.

name (input) The name of the View.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
cell is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
cell is not the OID of a Cell.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no View with the specified name exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

If the View was not opened prior to this function call, it is opened automatically with access mode CFIDR_READ. If the open fails, the error return for this function will be the error return of the open function.

The function **cfidrObjectGetObjectType()** must be used on the returned object to determine whether it is a **cfidrEncapsulatedViewIdT** or a **cfidrNetlistViewIdT**.

REFERENCE

cfidrObjectGetObjectType()

5.2.4 cfidrCellGetOwner

DECLARATION

cfidrLibIdT cfidrCellGetOwner(
cfidrCellIdT cell,
cfidrErrorT *error)

DESCRIPTION

This function returns the Lib at the end of the *Owner* relationship for the Cell specified by *cell.* The *Owner* relationship is defined when an object is created and cannot be modified.

RETURN VALUE

The return value is a cfidrLibIdT referring to Lib at the end of the *Owner* relationship. If an error occurs, a Null OID is returned.

PARAMETERS

cell (input) The OID of the Cell whose *Owner* relationship is to be followed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
cell is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
cell is not the OID of a Cell.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

5.2.5 cfidrCellGetViews

cfidrViewsIdT cfidrCellGetViews(
cfidrCellIdT cell,
cfidrErrorT *error)

DESCRIPTION

This function initiates a traversal of all of the Views at the end of the *Views* relationship defined for the Cell specified by *cell.*

RETURN VALUE

The return value is a cfidrViewsIdT referring to an Iterator ID that iterates over Views. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the **cfidrIterNextView ()** function returns a Null OID.

PARAMETERS

cell (input) The OID of a Cell for which the *Views* relationship is to be traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
cell is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
cell is not the OID of a Cell.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The **cfidrIterNextView()** function returns the next View OID in the Views relationship. If the View was not opened prior to the call to **cfidrIterNextView()**, it is opened automatically (i.e., all views are iterated, whether they are opened or not). The AccessMode for the open will be CFIDR_READ. If the open fails, the error return for this function will be the error return of the open.

If no objects are currently present in the *Views* relationship, an error is not returned but the first call to **cfidrIterNextView()** returns a Null OID.

REFERENCE

cfidrIterNextView()

5.3 Instance Functions

5.3.1 cfidrInstCheckDescriber [2]

DECLARATION

```
cfidrBooleanT cfidrInstCheckDescriber(
cfidrInstIdT inst,
cfidrErrorT *error)
```

DESCRIPTION

This function checks for discrepancies between the Inst specified via *inst* and its *Describer* View. In addition to checking the Inst against the View, this function checks all of the PortInsts owned by the Inst against their *Describer* Ports.

Checking is performed on all PortInsts owned either directly or indirectly by inst (as returned by cfidrInstGetPortInsts with mode ITER_ALL).

[2] NOTE: This function can be written entirely using other DR-PI functions. The principal other functions required are: **cfidrInstGetDescriber(), cfidrInstGetPortInsts(), cfidrPortInstGetDescriber (), and cfidrNetlistViewGetPorts()**

RETURN VALUE

If no discrepancies are found between the Inst and the *Describer* View, CFIDR_FALSE is returned. Otherwise, CFIDR_TRUE is returned and the *error* output argument will denote the discrepancy.

PARAMETERS

inst (input) The OID of the Inst to check against its *Describer* View.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
inst is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
inst is not the OID of an Inst.

CFIDR_DESCRIBER_VIEW_NOT_FOUND:
could not find or access the *Describer* View for inst.

CFIDR_DESCRIBER_PORT_NOT_FOUND:
at least one PortInst on the inst does not have a corresponding Port on the inst *Describer* View.

CFIDR_NEW_DESCRIBER_PORT_FOUND:
there is at least one additional Port on the inst *Describer* View that does not correspond to any PortInst on inst.

CFIDR_PORT_MEMBERS_DIFFER:
the Members of at least one PortInstBundle differ from the Members of the corresponding PortBundle. This code is also used if the Describer Port is a Scalar but the PortInst is a Bundle or the Describer is a Bundle, but the PortInst is a Scalar. This error is also returned if the owner of the describer of a PortInst is not the same object as the describer of the owner of that PortInst.

CFIDR_INTERNAL_SYSTEM_ERROR:

some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

If the *Describer* View is not opened yet, then this function will attempt to open it. The AccessMode for the open will be CFIDR_READ. If the open fails, the error return for this function will be the error return of the open

REFERENCE

cfidrInstGetDescriber()
cfidrInstGetPortInsts()
cfidrPortInstGetDescriber()
cfidrNetlistViewGetPorts()
cfidrPIOpenView()

5.3.2 cfidrInstFindPortInstByDescriberName

DECLARATION

cfidrPortInstIdT cfidrInstFindPortInstByDescriberName(
cfidrInstIdT inst,
cfidrStringT describerName,
cfidrErrorT *error)

DESCRIPTION

This function returns the PortInst (either PortInstBundle or PortInstScalar) with the specified describerName that is within the PortInsts relationship defined for the Inst specified by inst. All PortInsts are scoped to their Owner (either Inst or PortInstBundle), therefore PortInsts contained in PortInstBundles are not searched by this function.

RETURN VALUE

The return value is a cfidrPortInstIdT referring to the PortInst just found. If an error occurs, a Null OID is returned.

PARAMETERS

inst (input) The OID of the Inst from which the PortInst is to be found.

describerName (input) The *DescriberName* of the PortInst.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
inst is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
inst is not the OID of an Inst.

CFIDR_INVALID_NAME:
the describerName parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no PortInst with the specified describerName exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

5.3.3 cfidrInstGetDescriber

DECLARATION

cfidrNetlistViewIdT cfidrInstGetDescriber(
cfidrInstIdT inst,
cfidrErrorT *error)

DESCRIPTION

This function returns the View at the end of the *Describer* relationship for the Inst specified by *inst.*

RETURN VALUE

The return value is of type cfidrNetlistViewIdT and is the NetlistView at the end of the *Describer* relationship. If an error occurs, a Null OID is returned.

PARAMETERS

inst (input) The OID of the Inst whose *Describer* relationship is to be followed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
inst is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
inst is not the OID of an Inst.

CFIDR_DESCRIBER_VIEW_NOT_FOUND:
the *Describer* View for inst does not exist.

CFIDR_VIEW_OPEN_FAILED:
the *Describer* Netlist View could not be opened for some reason, such as lack of access rights.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

If the *Describer* View is not open, this function will attempt to open it. The AccessMode for the open will be CFIDR_READ.

REFERENCE

cfidrInstCheckDescriber()
cfidrPortInstGetDescriber()
cfidrPIOpenView().

5.3.4 cfidrInstGetOwner

DECLARATION

cfidrNetlistViewIdT cfidrInstGetOwner(
cfidrInstIdT inst,
cfidrErrorT *error)

DESCRIPTION

This function returns the View at the end of the *Owner* relationship for the Inst specified by *inst*. Note: The *Owner* relationship is defined when an object is created and cannot be modified.

RETURN VALUE

The return value is a cfidrNetlistViewIdT referring to the NetlistView at the end of the *Owner* relationship. If an error occurs, a Null OID is returned.

PARAMETERS

inst (input) The OID of the Instance whose *Owner* relationship is to be followed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
inst is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
inst is not the OID of an Inst.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

5.3.5 cfidrInstGetPortInsts

DECLARATION

cfidrPortInstsId cfidrInstGetPortInsts(
cfidrInstIdT inst,
cfidrIterModeT mode,
cfidrErrorT *error)

DESCRIPTION

This function initiates a traversal of all of the PortInsts (PortInstBundles, PortInstBusses, or PortInstScalars) at the end of the *PortInsts* relationship defined for the Inst specified by *inst*.

RETURN VALUE

The return value is a cfidrPortInstsIdT which is the ID of an Iterator that returns PortInsts. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextPortInst() function returns a Null OID.

PARAMETERS

inst (input) The OID of an Inst for which the *PortInsts* relationship is to be traversed.

mode (input) The mode that determines which PortInsts are returned. The current valid values are CFIDR_ITER_SCALARS, CFIDR_ITER_BUNDLES, CFIDR_ITER_TOP, and CFIDR_ITER_ALL. See the Pre-Conditions below for more detail on how these modes behave.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
inst is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:

inst is not the OID of an Inst.

CFIDR_INVALID_ITERMODE:
mode is not a valid IterMode.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The PortInsts returned depends on the *mode* argument specified.

mode=**CFIDR_ITER_TOP**
Returns all those PortInsts (PortInstBundles, PortInstBusses, or PortInstScalars) which are owned directly by the Inst (i.e., not owned by a PortInstBundle).

mode=**CFIDR_ITER_SCALARS**
Returns all the PortInstScalars owned directly or indirectly by the Inst. No PortInstBundles are output with this mode.

mode=**CFIDR_ITER_BUNDLES**
Returns all the PortInstBundles owned directly or indirectly by the Inst. No PortInstScalars are output with this mode.

mode=**CFIDR_ITER_ALL**
Returns all the PortInsts (PortInstBundles, PortInstBusses or PortInstScalars) owned directly or indirectly by the Inst. This is the union of all the PortInsts returned by the CFIDR_ITER_SCALARS and CFIDR_ITER_BUNDLES modes.

Depth-first recursion is used for all modes except CFIDR_ITER_TOP.

POST-CONDITIONS

The **cfidrIterNextPortInst()** function returns the next PortInst OID in the *PortInsts* relationship.

If no objects are currently present in the *PortInsts* relationship, an error is not returned, but the first call to **cfidrIterNextPortInst()** returns a Null OID.

The PortInst name scoping rules (inherited from the Port name scoping rules) state that PortInst names are only unique within their immediate owner (either an Inst or a PortInstBundle). Thus, two PortInsts with the same name may exist and be completely different objects. Thus, PortInsts returned via any mode except CFIDR_ITER_TOP may have the same name but in fact be different objects. The **cfidrObjectIsSame()** function can be used to determine if any two of these PortInsts are the same.

REFERENCE

cfidrIterNextPortInst()

cfidrObjectIsSame()

5.4 Iterator Functions

5.4.1 cfidrIterNextCell

DECLARATION

cfidrCellIdT cfidrIterNextCell(
cfidrCellsIdT iter,
cfidrErrorT *error)

DESCRIPTION

This function returns another Cell object from the Iterator ID specified via *iter*.

RETURN VALUE

The return value is a cfidrCellIdT referencing the Cell just iterated. If an error occurs or there are no more objects to iterate, a Null OID is returned. If there are no more objects to iterate, this is not an error.

PARAMETERS

iter (input) The Iterator ID representing an Iterator of Cell objects.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iter is not the ID of an Iterator of Cells.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *iter* argument must have been returned via a previous call to **cfidrLibGetCells()**.

REFERENCE

cfidrLibGetCells()

5.4.2 cfidrIterNextInst

DECLARATION

cfidrInstIdT cfidrIterNextInst(
cfidrInstsIdT iter,
cfidrErrorT *error)

DESCRIPTION

This function returns another Inst object from the Iterator ID specified via *iter*.

RETURN VALUE

The return value is a cfidrInstIdT referencing the Inst just iterated. If an error occurs or there are no more objects to iterate, a Null OID is returned. If there are no more objects to iterate, this is not an error.

PARAMETERS

iter (input) The Iterator ID representing an Iterator of Inst objects.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iter is not the ID of an Iterator of Insts.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *iter* argument must have been returned via a previous call to **cfidrNetlistViewGetInsts()**.

REFERENCE

cfidrNetlistViewGetInsts()

5.4.3 cfidrIterNextLib

DECLARATION

cfidrLibIdT cfidrIterNextLib(
cfidrLibsIdT iter,
cfidrErrorT *error)

DESCRIPTION

This function returns another Lib (library) object from the Iterator ID specified via *iter*.

RETURN VALUE

The return value is a cfidrLibIdT referencing the Lib object just iterated.

If an error occurs or there are no more Libs to iterate, a Null OID is returned. If there are no more Libs to iterate, this is not an error.

PARAMETERS

iter (input) The Iterator ID representing an Iterator of Lib objects.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iter is not the ID of an Iterator of Lib objects.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *iter* argument must have been returned via a previous call to cfidrPIGetLibs().

POST-CONDITIONS

If the Lib is not already open, this function will attempt to open it. The AccessMode for the open will be CFIDR_READ. If the open fails, the error return for this function will be the error return of the open. The iteration sequence can continue after a failed Lib open.

REFERENCE

cfidrPIGetLibs()

5.4.4 cfidrIterNextNet

DECLARATION

cfidrNetIdT cfidrIterNextNet(
cfidrNetsIdT iter,
cfidrErrorT *error)

DESCRIPTION

This function returns another Net object (NetBundle, NetBus, or NetScalar) from the Iterator ID specified via *iter*.

RETURN VALUE

The return value is a cfidrNetIdT referencing the Net just iterated. If an error occurs or there are no more objects to iterate, a Null OID is returned. If there are no more objects to iterate, this is not an error.

PARAMETERS

iter (input) The Iterator ID representing an Iterator of Net objects.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iter is not the ID of an Iterator of Nets.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *iter* argument must have been returned via a previous call to cfidrNetBundleGetNets() or cfidrNetlistViewGetNets().

POST-CONDITIONS

Nets in a NetBundle are ordered with Positions from 0 to one less than the number of nets in the bundle. When nets are being returned from a NetBundle they are returned in order of their Position. A net can be inserted in a Bundle at a particular position using by the **cfidrNetBundleInsertNet()** function which also causes all nets to the "right" of the newly inserted one to increase their Position by one. Net ordering is also affected by use of the **cfidrNetBundleRemoveNet()** function which causes all nets to the "right" of the removed net to decrease their position by one.

Nets being returned from the View are not returned in any guaranteed order.

REFERENCE

cfidrNetlistViewGetNets()
cfidrNetBundleInsertNet()
cfidrNetBundleRemoveNet()
cfidrNetBundleGetNets()
cfidrNetGetNetBundles()
cfidrPortBundleGetNetBundles()
cfidrPortInstBundleGetNetBundles()

5.4.5 cfidrIterNextNetBundle

DECLARATION

cfidrNetBundleIdT cfidrIterNextNetBundle(
cfidrNetBundlesIdT iter,
cfidrErrorT *error)

DESCRIPTION

This function returns another NetBundle object from the Iterator ID specified via *iter*.

RETURN VALUE

The return value is a cfidrNetBundleIdT referencing the NetBundle just iterated. If an error occurs or there are no more objects to iterate, a Null OID is returned. If there are no more objects to iterate, this is not an error.

PARAMETERS

iter (input) The Iterator ID representing an Iterator of NetBundle objects.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iter is not the ID of an Iterator of NetBundles.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *iter* argument must have been returned via a previous call to **cfidrNetGetNetBundles(), cfidrPortBundleGetNetBundles(), cfidrPortInstBundleGetNetBundles()**

REFERENCE

cfidrNetGetNetBundles()
cfidrPortBundleGetNetBundles()
cfidrPortInstBundleGetNetBundles()

5.4.6 cfidrIterNextPort

DECLARATION

cfidrPortIdT cfidrIterNextPort(
cidrPortsIdT iter,
cfidrErrorT *error)

DESCRIPTION

This function returns another Port object (PortBundle, PortBus, or PortScalar) from the Iterator ID specified via *iter*.

RETURN VALUE

The return value is a cfidrPortIdT referencing the Port just iterated. If an error occurs or there are no more objects to iterate, a Null OID is returned. If there are no more objects to iterate, this is not an error.

PARAMETERS

iter (input) The Iterator ID representing an Iterator of Port objects.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iter is not the ID of an Iterator of Ports.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *iter* argument must have been returned via a previous call to **cfidrNetlistViewGetPorts()**, **cfidrPortBundleGetPorts()**, or **cfidrNetGetPorts()**.

POST-CONDITIONS

Ports are returned in the order of their Position either in the View or in the PortBundle. Position is specified at creation time or using **cfidrPortSetOwnerView()** or **cfidrPortSetOwnerPortBundle()**.

REFERENCE

cfidrNetlistViewGetPorts()
cfidrPortBundleGetPorts()
cfidrPortSetOwnerPortBundle()
cfidrPortSetOwnerView()
cfidrNetGetPorts()

5.4.7 cfidrIterNextPortInst

DECLARATION

cfidrPortInstIdT cfidrIterNextPortInst(
cfidrPortInstsIdT iter,
cfidrErrorT *error)

DESCRIPTION

This function returns another PortInst object (PortInstBundle, PortInstBus, or PortInstScalar) from the Iterator ID specified via *iter*.

RETURN VALUE

The return value is a cfidrPortInstIdT referencing the PortInst just iterated. If an error occurs or there are no more objects to iterate, a Null OID is returned. If there are no more objects to iterate, this is not an error.

PARAMETERS

iter (input) The Iterator ID representing an Iterator of PortInst objects.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iter is not the ID of an Iterator of PortInsts.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *iter* argument must have been returned via a previous call to **cfidrInstGetPortInsts()**, **cfiPortInstBundleGetPortInsts()**, or **cfidrNetGetPortInsts()**.

REFERENCE

cfidrInstGetPortInsts()
cfidrPortInstBundleGetPortInsts()
cfidrNetGetPortInsts()

5.4.8 cfidrIterNextProp

DECLARATION

```
cfidrPropIdT cfidrIterNextProp(
cfidrPropsIdT iter,
cfidrErrorT *error)
```

DESCRIPTION

This function returns another Property object from the Iterator ID specified via *iter*.

RETURN VALUE

The return value is a cfidrPropIdT referencing the Property just iterated. If an error occurs or there are no more objects to iterate, a Null OID is returned. If there are no more objects to iterate, this is not an error.

PARAMETERS

iter (input) The Iterator ID representing an Iterator of Property objects.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iter is not the ID of an Iterator of Props.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *iter* argument must have been returned via a previous call to **cfidrObjectGetProps()**.

REFERENCE

cfidrObjectGetProps()

5.4.9 cfidrIterNextView

DECLARATION

```
cfidrViewIdT cfidrIterNextView(
cfidrViewsIdT iter,
cfidrErrorT *error)
```

DESCRIPTION

This function returns another View object from the Iterator ID specified via *iter*.

RETURN VALUE

The return value is a cfidrViewIdT referencing the View just iterated. If an error occurs or there are no more objects to iterate, a Null OID is returned. If there are no more objects to iterate, this is not an error.

PARAMETERS

iter (input) The Iterator ID representing an Iterator of View objects.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iter is not the ID of an Iterator of Views.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *iter* argument must have been returned via a previous call to **cfidrCellGetViews()**.

POST-CONDITIONS

If the View was not open prior to calling this function, it is opened automatically. The AccessMode for the open will be CFIDR_READ. If the open fails, the error return for this function will be the error return of the open. The iteration sequence can continue after a failed open.

The function **cfidrObjectGetObjectType()** must be used to determine whether each View is an EncapsulatedView or a NetlistView.

REFERENCE

cfidrCellGetViews()
cfidrObjectGetObjectType()

5.4.10 cfidrIterQuit

DECLARATION

```
cfidrVoidT cfidrIterQuit(
cfidrIterIdT iter,
cfidrErrorT *error)
```

DESCRIPTION

This function exits from a previously initialized Iterator specified via *iter*. This function is called once for each Iterator ID created. It is the application's responsibility for calling this function.

PARAMETERS

iter (input) The Iterator ID referring to the Iterator to be terminated.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

Once this function is called, trying to use the Iterator ID with any other function should result in an error.

5.4.11 cfidrIterReset

DECLARATION

```
cfidrVoidT cfidrIterReset(
cfidrIterIdT iter,
cfidrErrorT *error)
```

DESCRIPTION

This function resets the Iterator specified via *iter* such that it starts over from its beginning.

RETURN VALUE

None. If there is an error, calling the appropriate cfidrIterNext... function returns a Null OID.

PARAMETERS

iter (input) The Iterator ID referring to the generator to reset.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the OID of a valid Iterator.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The Iterator is reset such that calling the appropriate **cfidrIterNext...** function will iterate the first object.

NOTE: The order of the objects iterated after resetting the Iterator is guaranteed to be same as the previous iteration if and only if the relationship itself is ordered (is a List).

5.5 Lib (library) Functions

5.5.1 cfidrLibCreateCell

DECLARATION

cfidrCellIdT cfidrLibCreateCell(
cfidrLibIdT owner,
cfidrStringT name,
cfidrErrorT *error)

DESCRIPTION

This function creates a Cell object with its *Name* attribute set to *name* and owned by the Lib specified via *owner*.

RETURN VALUE

The return value is a cfidrCellIdT which refers to the Cell object just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the Lib object which owns the Cell.

name (input) The string representing the name of the Cell to create.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a Lib.

CFIDR_READ_ONLY:
the library has not been opened for update.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_ALREADY_EXISTS:
a Cell with the same name already exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The Lib specified by *owner* must previously have been created or opened for update.
The *name* argument must conform to the character set rules for name strings.

POST-CONDITIONS

A created Cell object is not persistent across session boundaries until the Lib owning the Cell is saved.

5.5.2 cfidrLibFindCellByName

DECLARATION

```
cfidrCellIdT cfidrLibFindCellByName(
cfidrLibIdT lib,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the Cell with the specified name that is within the Cells relationship defined for the Lib specified by lib.

RETURN VALUE

The return value is a cfidrCellIdT referring to the Cell just found. If an error occurs, a Null OID is returned.

PARAMETERS

lib (input) The OID of the Lib from which the Cell is to be found.

name (input) The name of the Cell.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
lib is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
lib is not the OID of a Lib.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no Cell with the specified name exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

5.5.3 cfidrLibGetCells

DECLARATION

```
cfidrCellsIdT cfidrLibGetCells(
cfidrLibIdT lib,
cfidrErrorT *error)
```

DESCRIPTION

This function initiates a traversal of all of the Cells at the end of the *Cells* relationship defined for the Lib specified by *lib*.

RETURN VALUE

The return value is a cfidrCellsIdT referring to an Iterator ID that iterates over Cells. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the **cfidrIterNextCell()** function returns a Null OID.

PARAMETERS

lib (input) The OID of the Lib for which the *Cells* relationship is to be traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
lib is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
lib is not the OID of a Lib.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The **cfidrIterNextCell()** function returns the next Cell OID in the *Cells* relationship.

If no objects are currently present in the *Cells* relationship, an error is not returned but the first call to **cfidrIterNextCell()** returns a Null OID.

REFERENCE

cfidrIterNextCell()

5.5.4 cfidrLibPurge

DECLARATION

```
cfidrVoidT cfidrLibPurge(
cfidrLibIdT lib,
cfidrErrorT *error)
```

DESCRIPTION

This function purges the Lib (library) specified via *lib*. The effect of purging a Lib is to also purge the entire Object hierarchy owned by the purged Lib.

PARAMETERS

lib (input) The OID of the Lib to purge.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
lib is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
lib is not the OID of a Lib

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

A Lib can be purged regardless of the access mode of any object.

POST-CONDITIONS

Any OIDs referring to purged Libs are made unusable. They may be reused to refer to other Libs at some point, so the application cannot assume that if the purged Lib is again made available (via open, create, etc.), that it will have the same OID.

Policy: The purge operation removes any changes made to an object since the last save or open. The object then becomes unavailable and its OID becomes unusable. Opening the Lib again assigns the Lib an OID and makes it available for updating.

An example implementation that would provide this behavior would make an in-memory copy of the Lib and its contained objects. All edits would be made only to the in-memory copy of the objects. The effect of a "purge" is to remove the in-memory copy of the objects, leaving the original disk copy of the objects unchanged.

Policy: Similar to **open**, **purge** also recursively walks the ownership hierarchy rooted by the specified Lib and purges all Objects in this hierarchy. Unlike **open**, **purge** continues walking the ownership hierarchy when it encounters an Object which can be purged individually. Thus, **purge** behaves similarly to **save** in this respect.

Rationale: Consider the information model and take into account the policy that only Libraries and Views can be purged. Purging a Lib not only purges the Lib and Cells, it also purges all the Views as well.

Rationale: When opening a Lib, opening all the Views automatically is probably undesirable in most situations. However, when purging a Lib, it does not make sense to only purge the Lib and Cells and not the Views.

REFERENCE

cfidrLibSave()
cfidrViewPurge()

5.5.5 cfidrLibSave

DECLARATION

cfidrVoidT cfidrLibSave(
cfidrLibIdT lib,
cfidrErrorT *error)

DESCRIPTION

This function saves the Lib (library) specified via *lib*. Saving a Lib makes all updates to objects in the Lib's hierarchy persistent. If the Lib has been opened for CFIDR_READ, changes to the Views contained in the Lib that are open for CFIDR_UPDATE will be made persistent. All Views owned by all Cells owned by *lib* will be also saved by this function.

RATIONALE

This function is a "brute force" save-everything-below-me function. Incremental saves to particular Views should be done using **cfidrViewSave()** which will only save whatever portions of the Lib are necessary to ensure referential integrity.

PARAMETERS

lib (input) The OID of the Lib to save. The Objects owned by *lib* are also saved.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
lib is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
lib is not the OID of a Lib.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Any View that is owned by a Cell owned by *lib* but which is NOT to be saved must first be purged with **cfidrViewPurge()** or destroyed with **cfidrObjectDestroy()** before calling **cfidrLibSave()**.

POST-CONDITIONS

All Views owned by all Cells owned by *lib* are now persistant and can later be retreived using **cfidrViewOpen()** or by **cfidrLibOpen()** followed by traversal of the Cells of the Lib and the Views of the Cells.

If Views owned by Cells owned by *lib* are created or destroyed after the execution of this function then

explicit calls to **cfidrViewSave()** can be used to update the Lib and ensure that it has Cells owning those views.

Once a Lib is saved, it is known and available for opening across session boundaries until destroyed via **cfidrObjectDestroy()**.

Saving does not invalidate OIDs referring to the saved Lib.

REFERENCE

cfidrObjectDestroy()
cfidrPIQuit()
cfidrViewPurge()
cfidrViewSave()

5.6 Named Object Functions

5.6.1 cfidrNamedObjectGetName

DECLARATION

cfidrStringT cfidrNamedObjectGetName(
cfidrNamedObjectIdT namedObject,
cfidrErrorT *error)

DESCRIPTION

This function returns the *Name* attribute of the NamedObject specified by *namedObject.*

RETURN VALUE

The return value is a cfidrStringT representing the *Name* of the object. On error the empty string "" is returned.

PARAMETERS

namedObject (input) The OID of the NamedObject whose *Name* attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
namedObject is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:

namedObject is not the OID of a NamedObject.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Note that PortInsts only have a *DescriberName* attribute which must be accessed via the **cfidrPortInstGetDescriberName()** function. There are other ViewSelection objects which do not have a Name attribute.

POST-CONDITIONS

The string returned is guaranteed to conform to the rules for name strings.

The memory for the string is managed by the DR-PI. The string's value remains valid until the next execution of any PI function which has a **cfidrStringT** return value or until the function **cfidrPIQuit()** is called.

REFERENCE

cfidrNamedObjectSetName()
cfidrPIQuit()
cfidrPortInstGetDescriberName()

5.6.2 cfidrNamedObjectSetName

DECLARATION

```
cfidrVoidT cfidrNamedObjectSetName(
cfidrNamedObjectIdT namedObject,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function sets the *Name* attribute of the NamedObject specified via *namedObject.*

PARAMETERS

namedObject (input) The OID of the NamedObject for which the *Name* attribute is set.

name (input) The string to set the *Name* attribute to. The caller is responsible for managing the memory of this string.

NOTE: DR-PI functions which return **cfidrStringT** values guarantee those strings to not be updated

until the next call to a PI function that returns another **cfidrStringT**. These strings can be used as the *name* argument because the **cfidrNamedObjectSetName()** function will not affect these strings returned from other PI functions.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
namedObject is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
namedObject is not the OID of a NamedObject.

NOTE: At the moment all objects (except PortInsts, ViewNameSelectors, ViewTypeSelectors, and Library Selectors) are namedObjects.

CFIDR_ILLEGAL_OPERATION:
namedObject is not allowed to have the name changed at this time.

CFIDR_READ_ONLY:
the containing object (Lib or View) has not been opened for update.

CFIDR_INVALID_NAME:
name parameter is not a valid string or is not a legal name.

CFIDR_NAME_IN_USE:
another NamedObject in the same name scope already is using the name name.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules that apply to all name strings. Any violation of the rules results in an error and the *Name* attribute not being set.

Note that all subtypes of cfidrNamedObject have a *Name* attributes. However, some of these names cannot be updated or have limitations as to when name changes are allowed. The behavior of each object type is:

Lib

Policy: The Lib's *Name* attribute cannot be set.
Rationale: Issues surround update of Insts referencing *Describer* Views.

Cell

Policy: The Cell's *Name* attribute cannot be set.
Rationale: Issues surround update of Insts referencing *Describer* Views.

View

Policy: The View's *Name* attribute cannot be set. To change a View's *Name,* it must be deleted and re-created.
Rationale: Issues surround update of Insts referencing *Describer* Views.

Port

Policy: A Port's *Name* attribute (either a PortBundle *Name* or PortScalar *Name)* cannot be set. To change a Port's *Name,* the Port must be deleted and re-created.
Rationale: Issues surround update of PortInsts referencing *Describer* Ports.

Net

Policy: A Net *Name* (either a NetBundle, NetBus, or NetScalar *Name*) can only be set when the Net is completely un-connected. This means that the Nets contained by the NetBundle must also be completely un-connected.
Rationale: Allowing the Net *Name* to be set when connected to Ports or PortInsts may affect the validity of those connections on some systems.

Policy: A Net's *Name* must be unique with respect to all other Nets owned by the same NetlistView.
Rationale: This follows from the name uniqueness and scoping rules for NetScalars.

Inst

Policy: The *Name* attribute must be unique with respect to existing Insts owned by the NetlistView.

Property

Policy: The *Name* attribute must be unique with respect to existing Property objects owned by the Property's owner.

SelectorSets

The Name attribute must be unique with respect to existing SelectorSet objects.

ViewNameSelectors, ViewTypeSelectors, and LibrarySelectors are not NamedOjects..

POST-CONDITIONS

The updated object must be saved in order to be persistent. Purging the object before saving it will result in the loss of the changes.

REFERENCE

cfidrNamedObjectGetName()

5.7 Net Functions

5.7.1 cfidrNetAttachPort

DECLARATION

cfidrVoidT cfidrNetAttachPort(
cfidrNetIdT net,
cfidrPortIdT port,
cfidrAttachModeT mode,
cfidrErrorT *error)

DESCRIPTION

This function connects the specified *net* and *port*. If there are no errors, it adds *port* to the *Ports* attribute of *net* and sets the *NetScalar* attribute of *port* to *net*.

RATIONALE

This function only directly attaches Scalars. It also "power" connects members of Bundles while recursively traversing the bundle hierarchy. NetBundle to PortBundle connection is an indirect effect of connections of members of those bundles ultimately due to some underlying Scalar connection.

RETURN VALUE

No function return value but *error* indicates status or success of connection.

PARAMETERS

net (input) The OID of a Net (Scalar, Bundle, or Bus).

port (input) The OID of a Port (Scalar, Bundle, or Bus).

mode (input) The mode of attachment. Currently, the valid values are CFIDR_ATTACH_BY_ORDER and CFIDR_ATTACH_BY_NAME

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
net is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:

net is not a Net OID.

CFIDR_UNUSABLE_PORT_OID:
port is not a usable OID.

CFIDR_INVALID_PORT_TYPE:
port is not a Port OID.

CFIDR_INVALID_ATTACH_MODE:
mode is not a legal AttachMode.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_CROSS_VIEW_OPERATION:
net and port are not in the same view. They must be in the same view.

CFIDR_SCALAR_BUNDLE_ATTACHMENT:
One of net and port is a Scalar, but the other is a Bundles or Bus. PortScalars may not be attached to NetBundles or NetBusses. NetScalars may not be attached to PortBundles or PortBusses.

CFIDR_PORT_ALREADY_ATTACHED:
port is already attached to a net.

CFIDR_NO_NAME_MATCHES:
mode is BY_NAME but no names in the PortBundle match any names in the NetBundle or the PortScalar and NetScalar have different names.

NOTE: This is not necessarily an error---bundles can be connected if previous scalars were connected but no new scalars are connected when this occurs. If the bundles were not previously connected, they will remain un-connected when this occurs.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

net = **NetBundle,** *port* = **PortBundle**

Description: Attempts to attach the PortBundle to the NetBundle's *Ports* relationship (i.e., connects the PortBundle and NetBundle). This is done by downward recursively trying to attach the members until eventually scalars are attached. Then bundles are recursively attached upward whenever one or more members are attached. The *mode* argument must be either CFIDR_ATTACH_BY_NAME or CFIDR_ATTACH_BY_ORDER.

CFIDR_ATTACH_BY_ORDER:
Attaching "by order" takes the ordered list of Nets contained by the NetBundle (or NetBus) and attempts

to connect them one-by-one in increasing order of position to the Ports owned by the PortBundle (or PortBus). These connection attempts continue until one of the lists is exhausted. It is not an error for the lists to be of differing size.

If a given position in the NetBundle contains a NetBundle but the same position in the PortBundle contains a PortScalar or if the NetBundle contains a NetScalar while the PortBundle contains a PortBundle, an error code (CFIDR_SCALAR_BUNDLE_ATTACHMENT) is returned and no connections are made.

If at a given position there is a NetScalar and a corresponding PortScalar then an attempt is made to connect them. That is the Id of the NetScalar becomes the value of the *NetScalar* attribute of the PortScalar (if there was not already another NetScalar connected) and the Id of the PortScalar is added to the set of PortScalars on the *PortScalars* attribute of the NetScalar. If the PortScalar is already connected (i.e., its *NetScalar* attribute is not Null) then an error code (CFIDR_PORT_ALREADY_ATTACHED) is returned and no connections are made.

If at a given position there is a Netbundle and a corresponding PortBundle then **cfidrNetAttachPort()** is recursively called with that NetBundle and PortBundle and *mode*=CFIDR_ATTACH_BY_ORDER. Note that recursive connections are "depth-first."

Either recursive calls are successful or no connections are made.

CFIDR_ATTACH_BY_NAME:
Attaching "by name" takes each Port owned by the PortBundle and attempts to connect it to the first Net contained in the NetBundle which has the same Name attribute. If a corresponding Name is not found then this Port is not connected and the next Port in the list is considered. If the corresponding Name is found but the type is incompatible (i.e., one is a Scalar but the other is a Bundle or Bus) an error (CFIDR_SCALAR_BUNDLE_ATTACHMENT) is returned and no connections are made.

If the corresponding *Name* is found and it is for a NetScalar and a PortScalar then an attempt is made to connect them. Connecting them means that the Id of the NetScalar becomes the value of the *NetScalar* attribute of the PortScalar (if there was not already another NetScalar connected) and the Id of the PortScalar is added to the set of PortScalars on the *PortScalars* attribute of the NetScalar. However, if the PortScalar is already connected (i.e., its *NetScalar* attribute is not Null) then an error code (CFIDR_PORT_ALREADY_ATTACHED) is returned and no connections are made.

If, for a given *Name*, there is both a Netbundle and a corresponding PortBundle then **cfidrNetAttachPort()** is recursively called with that NetBundle and PortBundle and *mode*=CFIDR_ATTACH_BY_NAME. Note that these recursive connections are "depth-first."

Either all recursive calls are successful or no connections are made.

Attaching a NetScalar to a PortScalar also implicitly connects the NetBundle containing the NetScalar to the PortBundle owning the PortScalar (i.e, the NetBundle's *Ports* attribute has the PortBundle added to it). This implicit connection is made for each NetBundle that contains the NetScalar---remembering that a NetScalar can belong to many NetBundles.

Policy: A PortBundle is attached to a NetBundle if and only if at least one PortScalar (in the PortBundle) and NetScalar (in the NetBundle) are attached.
Rationale: By creating this static requirement in the information model, bundle attachment is defined

unambiguously.

Policy: None of the PortScalars owned by the PortBundle can be already connected to any other NetScalar.
Rationale: a PortScalar connects to at most one NetScalar.

net = **NetScalar**, *port* = **PortScalar**

Description: If there are no errors: sets the *NetScalar* attribute of *port* to *net* and adds *port* to the *net*'s *Ports* attribute (i.e., connects the PortScalar and NetScalar). The *mode* argument has the following meaning: BY_NAME means the connection occurs only if the *net* and *port* have the same name. BY_ORDER allows the connection regardless of name assuming no other rules are violated. Note that multiple PortScalars on a NetScalar means that the PortScalars are "shorted" and at the next level up in the hierarchy NetScalars will be aliased as a result of the connection between the corresponding PortInstScalars.

Policy: The PortScalar cannot be already connected to another NetScalar.

Policy: Attaching a PortScalar owned by the NetlistView to a NetScalar not contained by a NetBundle is allowed.

Policy: If the attachment is finally allowed between a Port and a Net that are each owned by a Bundle, then the owning PortBundle is added implicitly to the *Ports* relationship of each NetBundle containing the Net.

NOTE: to be consistent with this automatic connection of Bundles, it is also necessary that, if a new NetBundle is created which includes a Net that is already attached to a Port which is a member of a PortBundle, the NetBundle and PortBundle will also be automatically connected.

Rationale: This relationship denotes that at least one of the PortScalars owned by the PortBundle is connected to one of the NetScalars contained by the NetBundle.

POST-CONDITIONS

Depending upon the OIDs input and their ownership, implicit relationships may be created between PortBundles/PortScalars and NetBundles/NetScalars. The policies defined in the Pre-Conditions section outline all of these scenarios.

If *net* and *port* are Bundles, some members of either bundle may not be attached after the function is executed due to either name mismatches or there being more members in one Bundle than in the other. This is NOT an error condition.

The updated objects must be saved in order to be persistent. Purging the objects before saving them will result in the loss of the changes.

5.7.2 cfidrNetAttachPortInst

DECLARATION

```
cfidrVoidT cfidrNetAttachPortInst(
cfidrNetIdT net,
cfidrPortInstIdT portInst,
cfidrAttachModeT mode,
cfidrErrorT *error)
```

DESCRIPTION

This function connects the specified *net* and *portInst*. If there are no errors it adds *portInst* to the *PortInsts* attribute of *net* and either sets the *NetScalar* attribute of *portInst* to *net* when both are Scalar or adds *net* to the *NetBundles* attribute of *portInst* when both are Bundles (or Busses).

RATIONALE

This function only directly attaches Scalars. It also "power" connects members of Bundles while recursively traversing the bundle hierarchy. NetBundle to PortInstBundle connection is an indirect effect of connections of members of those bundles ultimately due to some underlying Scalar connection.

PARAMETERS

net (input) The OID of a Net to whose *PortInsts* relationship *portInst* is to be added to.

portInst (input) The OID of a PortInst to add to the *PortInsts* relationship.

mode (input) The mode of attachment. Currently, the valid values are CFIDR_ATTACH_BY_ORDER and CFIDR_ATTACH_BY_NAME.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
net is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
net is not a Net OID.

CFIDR_UNUSABLE_PORTINST_OID:
portInst is not a usable OID.

CFIDR_INVALID_PORTINST_TYPE:
portInst is not a PortInst OID.

CFIDR_INVALID_ATTACH_MODE:
mode is not a legal AttachMode.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_CROSS_VIEW_OPERATION:
net and portInst are not in the same view. They must be in the same view.

CFIDR_SCALAR_BUNDLE_ATTACHMENT:
One of net and portInst is a Scalar, but the other is a Bundles or Bus. PortInstScalars may not be attached to NetBundles or NetBusses. NetScalars may not be attached to PortInstBundles or PortInstBusses.

CFIDR_PORTINST_ALREADY_ATTACHED:
portInst is already attached to a net.

CFIDR_NO_NAME_MATCHES:
mode is BY_NAME but no names in the PortInstBundle match any names in the NetBundle or the PortInstScalar and NetScalar have different names.

NOTE: This is not necessarily an error—bundles can be connected if previous scalars were connected but no new scalars are connected when this occurs. If the bundles were not previously connected, they will remain unconnected when this occurs.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

net = **NetBundle,** *port* = **PortInstBundle**

Description: Attaches the PortInstBundle to the NetBundle's *PortInsts* relationship (i.e., connects the PortInstBundle and NetBundle). The *mode* argument must be either CFIDR_ATTACH_BY_NAME or CFIDR_ATTACH_BY_ORDER.

CFIDR_ATTACH_BY_ORDER:
Attaching "by order" takes the ordered list of Nets contained by the NetBundle (or NetBus) and attempts to connect them one-by-one in increasing order of position to the ordered list of PortInsts owned by the PortInstBundle (or PortInstBus). These connection attempts continue until one of the lists is exhausted. It is not an error for the lists to be of differing size.

If a given position in the NetBundle contains a NetBundle but the same position in the PortInstBundle contains a PortInstScalar or if the NetBundle contains a NetScalar while the PortInstBundle contains a PortInstBundle, an error code (CFIDR_SCALAR_BUNDLE_ATTACHMENT) is returned and no connections are made.

If at a given position there is a NetScalar and a corresponding PortInstScalar then an attempt is made to connect them. That is the Id of the NetScalar becomes the value of the *NetScalar* attribute of the PortInstScalar (if there was not already another NetScalar connected) and the Id of the PortInstScalar is added to the set of PortInstScalars on the *PortInstScalars* attribute of the NetScalar. If the PortInstScalar is already connected (i.e., its *NetScalar* attribute is not Null) then an error code

(CFIDR_PORT_ALREADY_ATTACHED) is returned and no connections are made.

If at a given position there is a Netbundle and a corresponding PortInstBundle then **cfidrNetAttachPortInst()** is recursively called with that NetBundle and PortInstBundle and *mode*=CFIDR_ATTACH_BY_ORDER. Note that recursive connections are "depth-first."

CFIDR_ATTACH_BY_NAME:
Attaching "by name" takes each PortInst owned by the PortInstBundle and attempts to connect it to the first Net contained in the NetBundle which has the same Name attribute. If a corresponding Name is not found then this PortInst is not connected and the next PortInst in the list is considered. If the corresponding Name is found but the type is incompatible (i.e., one is a Scalar but the other is a Bundle or Bus) an error (CFIDR_SCALAR_BUNDLE_ATTACHMENT) is returned and no connections are made.

If the corresponding *Name* is found and it is for a NetScalar and a PortInstScalar then an attempt is made to connect them. Connecting them means that the Id of the NetScalar becomes the value of the *NetScalar* attribute of the PortInstScalar (if there was not already another NetScalar connected) and the Id of the PortInstScalar is added to the set of PortInstScalars on the *PortInstScalars* attribute of the NetScalar. However, if the PortInstScalar is already connected (i.e., its *NetScalar* attribute is not Null) then an error code (CFIDR_PORT_ALREADY_ATTACHED) is returned and no connections are made.

If, for a given *Name*, there is both a Netbundle and a corresponding PortInstBundle then **cfidrNetAttachPortInst()** is recursively called with that NetBundle and PortInstBundle and *mode*= CFIDR_ATTACH_BY_NAME. Note that these recursive connections are "depth-first."

Attaching a NetScalar to a PortInstScalar also implicitly connects the NetBundle containing the NetScalar to the PortInstBundle owning the PortInstScalar (i.e, the NetBundle's *PortInsts* attribute has the PortInstBundle added to it). This implicit connection is made for each NetBundle that contains the NetScalar---remembering that a NetScalar can belong to many NetBundles.

Policy: A PortInstBundle is attached to a NetBundle if and only if at least one PortInstScalar (in the PortInstBundle) and NetScalar (in the NetBundle) are attached.
Rationale: By creating this static requirement in the information model, bundle attachment is defined unambiguously.

Policy: None of the PortInstScalars owned by the PortInstBundle can be already connected to any other NetScalar.
Rationale: a PortInstScalar connects to at most one NetScalar.

net = **NetScalar,** *port* = **PortInstScalar**

Description: If there are no errors: sets the *NetScalar* attribute of *port* to *net* and adds *port* to the *net*'s *PortInsts* attribute (i.e., connects the PortInstScalar and NetScalar). The *mode* argument has the following meaning: CFIDR_ATTACH_BY_NAME means the connection occurs only if the *net* and *port* have the same name. CFIDR_ATTACH_BY_ORDER allows the connection regardless of name, assuming no other rules are violated.

Policy: The PortInstScalar cannot be already connected to another NetScalar.

Policy: Attaching a PortInstScalar to a NetScalar not contained by a NetBundle is allowed.

Policy: If the attachment is finally allowed between a PortInst and a Net that are each owned by a Bundle, then the owning PortInstBundle is added implicitly to the *PortInsts* relationship of each NetBundle containing the Net.

NOTE: To be consistent with this automatic connection of Bundles, it is also necessary that, if a new NetBundle is created which includes a Net that is already attached to a PortInst which is a member of a PortInstBundle, the NetBundle and PortInstBundle will also be automatically connected.

Rationale: This relationship occurs if at least one of the PortInsts owned by the PortInstBundle is connected to one of the Nets contained by the NetBundle.

POST-CONDITIONS

Depending upon the OIDs input and their ownership, implicit relationships may be created between PortInstBundles/PortInstScalars and NetBundles/NetScalars. The policies defined in the Pre-Conditions section outline all of these scenarios.

If *net* and *port*Inst are Bundles, some members of either bundle may not be attached after the function is executed due to either name mismatches or there being more members in one Bundle than in the other. This is NOT an error condition.

The updated objects must be saved in order to be persistent. Purging the objects before saving it will result in the loss of the changes.

5.7.3 cfidrNetBundleFindNetByName

DECLARATION

```
cfidrNetIdT cfidrNetBundleFindNetByName(
cfidrNetBundleIdT netBundle,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the Net with the specified name if it is within the Nets relationship defined for the NetBundle specified by netBundle.

RETURN VALUE

The return value is a cfidrNetIdT referring to the Net just found. If an error occurs, a Null OID is returned.

PARAMETERS

netBundle (input) The OID of the NetBundle in which the Net is to be found.

name (input) The name of the Net.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netBundle is not the OID of a NetBundle.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no Net with the specified name exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Only one level down in the specified *netBundle* is considered when searching for the Net with the specified *name*. Even if there is a Net elsewhere in the NetlistView (including in a lower level bundle) with the specified *name* it will only be returned by this function if it is in the specific *netBundle*. If the Net is to be returned regardless of which (if any) NetBundle contains it, the function **cfidrNetlistViewFindNetByName()** should be used.

POST-CONDITIONS

Note that a Net can appear at more than one position in a given NetBundle but since this is the exact same net regardless of position, it is meaningless to ask which position is returned.

REFERENCE

cfidrNetlistViewFindNetByName()

5.7.4 cfidrNetBundleFindNetByPosition

cfidrNetIdT cfidrNetBundleFindNetByPosition(
cfidrNetBundleIdT netBundle,
cfidrInt32T position
cfidrErrorT *error)

DESCRIPTION

This function returns the Net at the specified *position* within the specified *netBundle*. The "leftmost" or first position is numbered 0 and the "rightmost" or last position is numbered one less than the size of the bundle.

RETURN VALUE

The return value is a cfidrNetIdT referring to the Net just found. If an error occurs, a Null OID is returned.

PARAMETERS

netBundle (input) The OID of the NetBundle being accessed.

position (input) The position of the returned Net in the NetBundle. A value of 0 will return the Net at the first position. A value one less than the size of the bundle will return the Net at the last position. A value greater than or equal to the size of the bundle or any negative value will cause an error.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netBundle is not the OID of a NetBundle.

CFIDR_INVALID_POSITION:
the *position* parameter is not in the range 0 to one less than the size of the bundle.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Position of a Net in a NetBundle is determined by the use of the function **cfidrNetBundleInsertNet()** for this net and subsequently any nets at lower positions (to the left of it). It will also be affected by the use of the function **cfidrNetBundleRemoveNet()** on any nets at lower positions.

REFERENCE

cfidrNetBundleInsertNet()
cfidrNetBundleRemoveNet()

5.7.5 cfidrNetBundleGetNets

DECLARATION

```
cfidrNetsIdT cfidrNetBundleGetNets(
cfidrNetBundleIdT netBundle,
cfidrIterModeT mode,
cfidrErrorT *error)
```

DESCRIPTION

This function initiates a traversal of all of the Nets (NetBundles, NetBusses, or NetScalars) at the end of the *Nets* relationship defined for the NetBundle specified by *netBundle*. These are also known as the members of the NetBundle.

RETURN VALUE

The return value is a cfidrNetsIdT referring to an Iterator ID that iterates over Nets. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextNet() function returns a Null OID.

PARAMETERS

netBundle (input) The OID of an NetBundle for which the *Nets* relationship is to be traversed.

mode (input) The mode that determines which Nets are returned. The current valid values are CFIDR_ITER_SCALARS, CFIDR_ITER_BUNDLES, CFIDR_ITER_TOP, and CFIDR_ITER_ALL. See the Pre-Conditions below for more detail on how these modes behave.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netBundle is not the OID of a NetBundle.

CFIDR_INVALID_ITERMODE:
mode is not a valid IterMode.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Returns selected Nets contained by the NetBundle. Nets contained in a NetBundle need not be unique.

Thus, the same Net may be produced two or more times for any given NetBundle.

The order of the Nets on the Nets list of a given NetBundle is determined by the order and positions in which Nets are inserted into that NetBundle as well as by the removal of Nets from that NetBundle. See the functions cfidrNetBundleInsertNet() and cfidrNetBundleRemoveNet().

The specific Nets returned depend on the *mode* argument specified:

mode=CFIDR_ITER_TOP
Returns the Nets directly contained in *netBundle*.

mode=CFIDR_ITER_SCALARS
Returns all the NetScalars contained directly or recursively in *netBundle*, whether they are also contained in another NetBundle. No NetBundles are output with this mode, only NetScalars. This mode lets the application effectively see *netBundle* flattened to a list of NetScalars "below it".

mode=CFIDR_ITER_BUNDLES
Returns all the NetBundles contained directly or recursively in *netBundle*. No NetScalars are output.

mode=CFIDR_ITER_ALL
Returns all the Nets (NetBundles, NetBusses, or NetScalars) contained directly or recursively in *netBundle*. This mode differs from CFIDR_ITER_TOP in that NetScalars contained in a NetBundle are also returned. This mode is a combination of the SCALARS and BUNDLES modes.

Depth-first recursion is used for all modes except CFIDR_ITER_TOP, which uses no recursion.

POST-CONDITIONS

The **cfidrIterNextNet()** function returns the next Net OID in the *Nets* relationship. The next Net returned must obey the return mode as specified via the *mode* argument.

If no objects are currently present in the *Nets* relationship given the specified *mode*, an error is not returned but the first call to **cfidrIterNextNet()** returns a Null OID.

REFERENCE

cfidrIterNextNet()
cfidrNetBundleInsertNet()
cfidrNetBundleRemoveNet()
cfidrNetlistViewCreateNetBundle()

5.7.6 cfidrNetBundleGetSize

cfidrInt32T cfidrNetBundleGetSize(
cfidrNetBundleIdT netBundle,
cfidrErrorT *error)

DESCRIPTION

This function returns the number of Nets currently in the specified *netBundle*. This is only the number at the next level and not a count of the total number of scalars including those indirectly in this bundle. This is not the number of unique nets; if a Net is in the bundle more than once, each position of that Net contributes to the total size.

RETURN VALUE

The return value is a cfidrInt32T. If an error occurs, a -1 is returned.

PARAMETERS

netBundle (input) The OID of a NetBundle.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netBundle is not the OID of a NetBundle.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

NetBundles are created with size 0. Members are added via the **cfidrNetBundleInsertNet()** function and removed with the **cfidrNetBundleRemoveNet()** function.

REFERENCE

cfidrNetBundleInsertNet()
cfidrNetBundleRemoveNet()
cfidrNetlistViewCreateNetBundle()

5.7.7 cfidrNetBundleInsertNet

DECLARATION

```
cfidrVoidT cfidrNetBundleInsertNet(
cfidrNetBundleIdT netBundle,
cfidrNetIdT net,
cfidrInt32T position,
```

cfidrErrorT *error)

DESCRIPTION

This function inserts the existing *net* (NetBundle, NetBus, or NetScalar) in the NetBundle specified by netBundle at the position specified by position.

PARAMETERS

netBundle (input) The OID of the NetBundle into whose list of Nets *net* is being inserted.

net (input) The OID of the Net to be inserted into the NetBundle.

position (input) The desired location of the Net in the NetBundle. If position is less than 0 or greater than or equal to the number of members in the list of Nets in the NetBundle netBundle, the new Net is appended as the last member of the list. Otherwise, it is inserted at the specified position in the list (0 is the first or "leftmost" position).

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netBundle is not the OID of a NetBundle or a Net Bus.

CFIDR_UNUSABLE_NET_OID:
net is not a usable OID.

CFIDR_INVALID_NET_TYPE:
the net OID refers to an OID of an inappropriate type.

CFIDR_CROSS_VIEW_OPERATION:
the NetBundle netBundle and the Net net are owned by different Views.

CFIDR_MEMBER_RECURSION:
the netBundle and net OIDs refer to the same object.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The NetBundle netBundle and Net *net* to be added must both be owned by the same View.

The netBundle and net OIDs cannot refer to the same object since this would result in recursion. Checking for recursion via multiple levels of Bundle containment is encouraged but not required.

POST-CONDITIONS

The Net is added to the list of Nets owned by netBundle at position. If position=0, the new Net will be added as the first member of the list. If position is less than 0 or greater than or equal to the number of members in the list, the Net is appended to the end of the list. Otherwise, the Net is inserted at the specified position (0 based). The position of all members whose positions were greater than or equal to position, will have their position incremented by 1.

For example, presume a NetBundle has members:
A, B, C0, 1, 2.

Adding a new member X at position 1 results in:
A, X, B, C0, 1, 2

Note that adding X at position 0 is inserting at the beginning:
X, A, B, C0, 1, 2

and adding at an invalid position such as -1 causes the new member to be appended at the end:
A, B, C0, 1, 2, X

REFERENCE

cfidrNetBundleRemoveNet()
cfidrNetBundleGetNets()
cfidrNetlistViewCreateNetBundle()
cfidrNetlistViewGetNets()

5.7.8 cfidrNetBundleRemoveNet

DECLARATION

```
cfidrVoidT cfidrNetBundleRemoveNet(
cfidrNetBundleIdT netBundle,
cfidrInt32T position,
cfidrErrorT *error)
```

DESCRIPTION

This function removes a Net at position from a NetBundle (or NetBus).

PARAMETERS

netBundle (input) The OID of the parent NetBundle (or NetBus).

position (input) The location of the Net to be removed from *netBundle*. If *position* is less than 0 or greater than or equal to the size of the list of Nets in netBundle, an error is returned. Otherwise, the net at the specified position is removed (0 is the first member).

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netBundle is not the OID of a NetBundle or NetBus.

CFIDR_INVALID_POSITION:
the position specified is less than 0 or greater than or equal to the size of the list of Nets in the NetBundle. Note that legal positions range from 0 to
size-1.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The specified position must identify a net in the NetBundle netBundle.

POST-CONDITIONS

The Net at the specified position in the NetBundle will be removed from the list of members of netBundle. It is not deleted from the View or removed from membership in any other NetBundles (or NetBusses) or from any other position within this NetBundle.

All other members of the NetBundle with position values greater than the value held by the Net at *position* will have their position values decremented by one. That is, ordering of the remaining members Nets within the NetBundle will be preserved and the *positions* will start from zero and monotonically increase by one with no gaps.

For example, presume a NetBundle has members:
A, B, C0, 1, 2.
at positions:
0, 1, 2, 3, 4

Removing the member at position=1 (i.e., "B") results in:
A, C0, 1, 2
at positions:
0, 1, 2, 3

REFERENCE

cfidrNetBundleInsertNet()
cfidrNetBundleGetNets()
cfidrNetlistViewCreateNetBundle()
cfidrNetlistViewGetNets()

5.7.9 cfidrNetBusGetStart

DECLARATION

cfidrInt32T cfidrNetBusGetStart(
cfidrNetBusIdT netBus,
cfidrErrorT *error)

DESCRIPTION

This function returns the *start* attribute of the specified *netBus*. This specifies the first index value for this bus.

RETURN VALUE

The return value is a cfidrInt32T. If an error occurs, a 0 is returned.

PARAMETERS

netBus (input) The OID of a NetBus.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netBus is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netBus is not the OID of a NetBus.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The value of the Start attribute of a NetBus is determined at creation time or via the **cfidrNetBusSetStart()** function.

REFERENCE

cfidrNetBusSetStart()
cfidrNetlistViewCreateNetBus()

5.7.10 cfidrNetBusGetStep

DECLARATION

cfidrInt32T cfidrNetBusGetStep(
cfidrNetBusIdT netBus,
cfidrErrorT *error)

DESCRIPTION

This function returns the *Step* attribute of the specified *netBus*. This specifies the numeric increment between positions for the index.

RETURN VALUE

The return value is a cfidrInt32T. If an error occurs, a 0 is returned.

PARAMETERS

netBus (input) The OID of a NetBus.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netBus is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netBus is not the OID of a NetBus.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The value of the Step attribute of a NetBus is determined at creation time or by the use of the **cfidrNetBusSetStep()** function.

REFERENCE

cfidrNetBusSetStep()

5.7.11 cfidrNetBusSetStart

DECLARATION

```
cfidrVoidT cfidrNetBusSetStart(
cfidrNetBusIdT netBus,
cfidrInt32T start,
cfidrErrorT *error)
```

DESCRIPTION

This function sets the *Start* attribute of the specified *netBus* to *start*. This specifies the index value for position 0 in this bus.

PARAMETERS

netBus (input) The OID of a NetBus.

start (input) The 32 bit integer number for position 0 which is the first (leftmost) index of the bus.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netBus is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netBus is not the OID of a NetBus.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The index of position 0 is now *start*.

REFERENCE

cfidrNetBusGetStart()
cfidrNetlistViewCreateNetBus()

5.7.12 cfidrNetBusSetStep

DECLARATION

cfidrVoidT cfidrNetBusSetStep(
cfidrNetBusIdT netBus,
cfidrInt32T step,
cfidrErrorT *error)

DESCRIPTION

This function sets the *Step* attribute of the specified *netBus* to *step*. This specifies the numeric increment between positions by which the index changes.

PARAMETERS

netBus (input) The OID of a NetBus.

step (input) The 32 bit integer number for the step. This number must be non-zero.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netBus is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netBus is not the OID of a NetBus.

CFIDR_INVALID_VALUE:
step is zero.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The difference in index values of positions n and n+1 will be *step* after the execution of this function.

REFERENCE

cfidrNetBusGetStep()
cfidrNetlistViewCreateNetBus()

5.7.13 cfidrNetDetachPort

DECLARATION

cfidrVoidT cfidrNetDetachPort(
cfidrNetIdT net,
cfidrPortIdT port,
cfidrErrorT *error)

DESCRIPTION

This function detaches (removes) the Port specified via *port* from the *Ports* relationship defined for the Net specified via *net.*

PARAMETERS

net (input) The OID of a Net from whose *Ports* relationship *port* is to be removed from.

port (input) The OID of a Port to remove from the *Ports* relationship.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
net is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
net is not the OID of a Net.

CFIDR_UNUSABLE_PORT_OID:
port is not a usable OID.

CFIDR_INVALID_PORT_TYPE:
port is not a Port OID.

CFIDR_PORT_NOT_ATTACHED:
port is not attached to net. This code is also returned when net is a bundle but port is a scalar or net is a scalar and port is a bundle since these can never be attached.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

net = **NetBundle,** *port* = **PortBundle**

Description: Detaches (removes) the PortBundle from the NetBundle's *Ports* relationship (i.e., disconnects the PortBundle and NetBundle). Likewise, any PortScalars owned by the PortBundle are detached from their connected NetScalar's *Ports* relationship. See the discussion, "NetBundle Connection to Port[Inst]Bundle is a Derived Relationship" on page 4-35, for additional implications.

NOTE: The only PortScalars detached are those connected to NetScalars which are contained in the NetBundle specified via *net.* Other PortScalars owned by the PortBundle specified via *port* may be connected to NetScalars not contained in the NetBundle specified via *net.*

Policy: A PortBundle remains attached to a NetBundle if and only if at least one PortScalar (in the PortBundle) and NetScalar (in the NetBundle) are attached.

Rationale: By creating this static requirement in the information model, bundle attachment is defined unambiguously.

net = **NetScalar,** *port* = **PortScalar**

Description: Detaches (removes) the PortScalar from the NetScalar's *Ports* relationship.

Policy: If both the NetScalar and PortScalar are owned by appropriate Bundles, the PortBundle is likewise detached from the NetBundle if this is the last Scalar attachment being removed between the Bundles.

Rationale: There is no need to keep the NetBundle/PortBundle relationship around since it is automatically created if and when the Scalars attach again.

POST-CONDITIONS

Several relationships may be implicitly removed depending upon the input arguments.The policies defined in the Pre-Conditions section outline all of these scenarios.

The updated objects must be saved in order to be persistent. Purging the objects before saving it will result in the loss of the changes.

See the discussion, "NetBundle Connection to Port[Inst]Bundle is a Derived Relationship" on page 4-35, for additional implications.

REFERENCE

cfidrNetAttachPort().

5.7.14 cfidrNetDetachPortInst

DECLARATION

```
cfidrVoidT cfidrNetDetachPortInst(
cfidrNetIdT net,
cfidrPortInstIdT portInst,
cfidrErrorT *error)
```

DESCRIPTION

This function detaches (removes) the PortInst specified via *portInst* from the *PortInsts* relationship defined for the Net specified via *net.*

PARAMETERS

net (input) The OID of a Net from whose *PortInsts* relationship *portInst* is to be removed.

portInst (input) The OID of a PortInst to remove from the *PortInsts* relationship.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
net is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
net is not the OID of a Net.

CFIDR_UNUSABLE_PORTINST_OID:
portInst is not a usable OID.

CFIDR_INVALID_PORTINST_TYPE:
portInst is not a PortInst OID.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_PORTINST_NOT_ATTACHED:
portInst is not attached to net. This code is also returned when net is a bundle but port is a scalar or net is a scalar and port is a bundle since these can never be attached.

CFIDR_INTERNAL_SYSTEM_ERROR:

some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

***net* = NetBundle, *portInst* = PortInstBundle**

Description: Detaches (removes) the PortInstBundle from the NetBundle's *PortInsts* relationship (i.e., disconnects the PortInstBundle and NetBundle). Likewise, any PortInstScalars owned by the PortInstBundle are detached from their connected NetScalar's *PortInsts* relationship.

NOTE: The only PortInstScalars detached are those connected to NetScalars which are contained in the NetBundle specified via *net*. Other PortInstScalars owned by the PortBundle specified via *portInst* may be connected to NetScalars not contained in the NetBundle specified via *object*.

Policy: A PortInstBundle remains attached to a NetBundle if and only if at least one PortInstScalar (in the PortInstBundle) and NetScalar (in the NetBundle) are attached.
Rationale: By creating this static requirement in the information model, bundle attachment is defined unambiguously.

***net* = NetScalar, *portInst* = PortInstScalar**

Definition: Detaches (removes) the PortInstScalar from the NetScalar's *PortInsts* relationship.

Policy: If both the NetScalar and PortInstScalar are owned by appropriate Bundles, the PortInstBundle is likewise detached from the NetBundle if this is the last Scalar attachment being removed between the Bundles.
Rationale: There is no need to keep the NetBundle/PortInstBundle relationship around since it is automatically created if and when the Scalars attach again.

POST-CONDITIONS

Several relationships may be implicitly removed depending upon the input arguments.The policies defined in the PRE-CONDITIONS section outline all of these scenarios.

The updated objects must be saved in order to be persistent. Purging the objects before saving will result in the loss of the changes.

See the discussion, "NetBundle Connection to Port[Inst]Bundle is a Derived Relationship" on page 4-35, for additional implications.

REFERENCE

cfidrNetAttachPortInst()

5.7.15 cfidrNetFindNetBundleByName

DECLARATION

cfidrNetBundleIdT cfidrNetFindNetBundleByName(
cfidrNetIdT net,
cfidrStringT name,
cfidrErrorT *error)

DESCRIPTION

This function returns the NetBundle with the specified name that is within the NetBundles relationship defined for the Net specified by net.

RATIONALE

This is a search by name of all the NetBundles of which this Net is a member, a set that can be accessed using the **cfidrNetGetNetBundles()** function. This is possible because a Net can be in more than one NetBundle.

RETURN VALUE

The return value is a cfidrNetBundleIdT referring to the NetBundle just found. If an error occurs, a Null OID is returned.

PARAMETERS

net (input) The OID of the Net from which the NetBundle is to be found.

name (input) The name of the NetBundle.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
net is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
net is not the OID of a Net.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no NetBundle with the specified name exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:

no error occurred.

REFERENCE

cfidrNetGetNetBundles()

5.7.16 cfidrNetGetNetBundles

DECLARATION

cfidrNetBundlesIdT cfidrNetGetNetBundles(
cfidrNetIdT net,
cfidrErrorT *error)

DESCRIPTION

This function initiates a traversal of all of the NetBundles at the end of the *NetBundles* relationship defined for the Net specified by *net*. A Net can belong to more than one NetBundle. This function returns all the NetBundles of which *net* is a *Member*.

RETURN VALUE

The return value is a cfidrNetBundlesIdT referring to an Iterator ID that iterates over NetBundles. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextNetBundle() function returns a Null OID.

PARAMETERS

net (input) The OID of a Net for which the *NetBundles* relationship is to be traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
net is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
net is not the OID of a Net.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The **cfidrIterNextNetBundle()** function returns the next NetBundle OID in the *NetBundles* relationship.

If no objects are currently present in the *NetBundles* relationship, an error is not returned but the first call to **cfidrIterNextNetBundle()** returns a Null OID.

REFERENCE

cfidrIterNextNetBundle()

5.7.17 cfidrNetGetOwner

DECLARATION

```
cfidrNetlistViewIdT cfidrNetGetOwner(
cfidrNetIdT net,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the View at the end of the *Owner* relationship for the Net specified by *net.* The *Owner* relationship is defined when an object is created and cannot be modified.

RETURN VALUE

The return value is a cfidrNetlistViewIdT referring the NetlistView at the end of the *Owner* relationship. If an error occurs, a Null OID is returned.

PARAMETERS

net (input) The OID of the Net whose *Owner* relationship is to be followed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
net is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
net is not the OID of a Net

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

5.7.18 cfidrNetGetPorts

DECLARATION

cfidrPortsIdT cfidrNetGetPorts(
cfidrNetIdT net,
cfidrErrorT *error)

DESCRIPTION

This function initiates a traversal of all of the Ports (PortBundles, PortBusses, or PortScalars) at the end of the *Ports* relationship defined for the Net specified by *net*.

RETURN VALUE

The return value is a cfidrPortsIdT referring to an Iterator ID that iterates over Ports. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextPort() function returns a Null OID.

PARAMETERS

net (input) The OID of an object for which the *Ports* relationship is to be traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
net is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
net is not the OID of a Net.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *net* parameter may either be a NetScalar, NetBundle, or Netbus. The NetBundle and NetBus behave the same for this function.

NetScalar

Description: Returns all PortScalars connected by this NetScalar. Note that multiple PortScalars on a NetScalar means that the PortScalars are "shorted" and at the next level up in the hierarchy NetScalars will be aliased as a result.

NetBundle (or NetBus)

Description: Returns all PortBundles or PortBusses which are connected to the NetBundle (or NetBus). See the **cfidrNetAttachPort()** function for how NetBundles and PortBundles may be connected. All PortBundles, which directly or indirectly contain any PortScalars connected to NetScalars owned directly or indirectly by *net*, are returned.

POST-CONDITIONS

The **cfidrIterNextPort()** function returns the next Port OID in the *Ports* relationship.

If no objects are currently present in the *Ports* relationship, an error is not returned but the first call to **cfidrIterNextPort()** returns a Null OID.

REFERENCE

cfidrIterNextPort()
cfidrNetAttachPort()

5.7.19 cfidrNetGetPortInsts

DECLARATION

cfidrPortInstsIdT cfidrNetGetPortInsts(
cfidrNetIdT net,
cfidrErrorT *error)

DESCRIPTION

This function initiates a traversal of all of the PortInsts (PortInstBundles, PortInstBusses, or PortInstScalars) at the end of the *PortInsts* relationship defined for the Net specified by *net*.

RETURN VALUE

The return value is a cfidrPortInstsIdT referring to an Iterator ID that iterates over PortInsts. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextPortInst() function returns a Null OID.

PARAMETERS

net (input) The OID of a Net for which the *PortInsts* relationship is to be traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
net is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
net is not the OID of a Net.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *net* parameter may either be a NetScalar, NetBundle, or Netbus. The NetBundle and NetBus behave the same for this function.

NetScalar

Description: Returns all PortInstScalars connected by this NetScalar.

NetBundle (or NetBus)

Description: Returns all PortInstBundles or PortInstBusses which are connected to this NetBundle (or NetBus). See the **cfidrNetAttachPortInst()** function for how NetBundles and PortInstBundles may be connected. All PortInstBundles, which directly or indirectly contain any PortInstScalars connected to NetScalars owned directly or indirectly by *net*, are returned.

POST-CONDITIONS

The **cfidrIterNextPortInst()** function returns the next PortInst OID in the *PortInsts* relationship.

If no objects are currently present in the *PortInsts* relationship, an error is not returned but the first call to **cfidrIterNextPortInst()** returns a Null OID.

REFERENCE

cfidrIterNextPortInst()
cfidrNetAttachPortInst()

5.7.20 cfidrNetScalarGetIsGlobal

DECLARATION

cfidrBooleanT cfidrNetScalarGetIsGlobal(
cfidrNetScalarIdT netScalar,
cfidrErrorT *error)

DESCRIPTION

This function returns the *IsGlobal* attribute for the NetScalar specified in *netScalar*.

RETURN VALUE

The return value is a cfidrBooleanT which specifies whether this NetScalar is a global net. All global nets with the same name are the same electrically common net although different OIDs may be used for them in different NetlistViews. If an error occurs CFIDR_FALSE is returned.

PARAMETERS

netScalar (input) The OID of the NetScalar whose *IsGlobal* attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netScalar is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netScalar is not the OID of a NetScalar

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The value of *IsGlobal* is CFIDR_FALSE when a NetScalar is newly created by **cfidrNetlistViewCreateNetScalar()**. It may be set by the function **cfidrNetScalarSetIsGlobal()**.

REFERENCE

cfidrNetlistViewCreateNetScalar()
cfidrNetScalarSetIsGlobal()

5.7.21 cfidrNetScalarSetIsGlobal

DECLARATION

```
cfidrVoidT cfidrNetScalarSetIsGlobal(
cfidrNetScalarIdT netScalar,
cfidrBooleanT value
cfidrErrorT *error)
```

DESCRIPTION

This function sets the *IsGlobal* attribute for the NetScalar specified in *netScalar* to the cfidrBooleanT

value specified in *value*.

PARAMETERS

netScalar (input) The OID of the NetScalar whose *IsGlobal* attribute is to be set.

value (input) The desired cfidrBooleanT value for the *IsGlobal* attribute of *netScalar*.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netScalar is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netScalar is not the OID of a NetScalar

CFIDR_INVALID_VALUE:
value is not a valid Boolean value.

CFIDR_READ_ONLY:
the containing object (Lib or View) has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The value of *IsGlobal* is CFIDR_FALSE when a NetScalar is newly created by **cfidrNetlistViewCreateNetScalar()**. It may have been set by previous calls to this function.

POST-CONDITIONS

The *IsGlobal* attribute of *netScalar* will be *value* after the execution of this function. That value will be persistent after a call to **cfidrViewSave()**. Any call to **cfidrNetScalarGetIsGlobal()** prior to another call to **cfidrNetScalarSetIsGlobal()** will return *value* unless a call to **cfidrViewPurge()** is done before any call to **cfidrViewSave()**.

All NetScalars with *IsGlobal*==CFIDR_TRUE and the same value for *Name* are considered to be the same electrically common scalar net but each NetlistView may return a different OID for its global NetScalar by that name. Furthermore, the function **cfidrObjectIsSame()** will not necessarily return the value CFIDR_TRUE when called with two NetScalars from different NetlistViews even though their *Names* are the same and both have *IsGlobal* set to CFIDR_TRUE.

REFERENCE

cfidrNetScalarGetIsGlobal()
cfidrObjectIsSame()
cfidrViewPurge()
cfidrViewSave()
cfidrNetlistViewCreateNetScalar()

5.8 Object Functions

5.8.1 cfidrObjectCreateProp

DECLARATION

cfidrPropIdT cfidrObjectCreateProp(
cfidrObjectIdT owner,
cfidrStringT name,
cfidrErrorT *error

DESCRIPTION

This function creates a Property object owned by the object specified via *owner*. The *Name* attribute is set to *name*.

RATIONALE

This function does not specify the Property's *ValueType* or *Value* attributes during creation. If this were available, there would be a function for each valueType. To minimize the number of functions, this one function to create a named Property with a default *Value* and *ValueType* is used. The cfidrPropSet {Boolean, Float32, Int32,String}Value() functions are used to set those values and types after the property is created.

RETURN VALUE

The return value is a cfidrPropIdT referring to the Property just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the object which will own the created Property. This OID can refer to any object, even another Property.

name (input) The string representing the name of the Property.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of an Object which can own properties.

NOTE: This is not possible at this time since all objects currently can own properties.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_ALREADY_EXISTS:
a prop with the same name already exists.

CFIDR_READ_ONLY:
the containing object (Lib or View) has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

POST-CONDITIONS

The property can later be accessed via its *Name* attribute by using the **cfidrObjectFindPropByName()** function.

REFERENCE

cfidrObjectFindPropByName()
cfidrObjectGetProps()

5.8.2 cfidrObjectDestroy

DECLARATION

cfidrVoidT cfidrObjectDestroy(
cfidrObjectIdT object,
cfidrErrorT *error)

DESCRIPTION

This function destroys the object specified via *object*. The effect of destroying an object is to also destroy the entire object hierarchy owned by the destroyed object.

PARAMETERS

object (input) The OID of the object to be destroyed. The object hierarchy owned by *object* is also destroyed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
object is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
object is not the OID of a destroyable object. These are only all the subtypes of PortInst at this time.

CFIDR_READ_ONLY:
the containing object (Lib or View) has not been opened for update. Note that the containing object of a View is its parent Lib; aView can always be destroyed if its parent Lib is open for update. A Lib or SelectorSet can always be destroyed.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Since the various subtypes (scalar, bundle, and bus) of PortInst objects are not created explicitly (they're defined automatically when the Inst is created), they cannot be destroyed explicitly. They can only be destroyed by destroying the Inst.

POST-CONDITIONS

See the discussion, "NetBundle Connection to Port[Inst]Bundle is a Derived Relationship" on page 4-35, for additional implications.

NetScalars

Destroying a NetScalar removes the NetScalar from each NetBundle that owns the NetScalar (preserving the order of the Nets remaining within the NetBundle) and then destroys the actual NetScalar object.

If a NetScalar being destroyed is attached to PortScalars or PortInstScalars then the NetScalar is first detached from those objects and then destroyed.

NetBundles (and NetBusses)

If a NetBundle (or NetBus) being destroyed is attached to any PortBundles (or PortBusses) or

PortInstBundles (or PortInstBusses) by virtue of having NetScalars in the NetBundle attached to PortScalars or to PortInstScalars those Port[Inst]Bundles will be detached from the NetBundle before it is destroyed.

Destroying a NetBundle does NOT destroy the Nets contained by the NetBundle because a NetBundle is merely a "collector" or "grouper" of Nets and not the owner of those Nets.

Destroying a NetBundle does NOT affect any attachment connectivity below it in a hierarchy; however, NetBundles above it may be detached if the destroyed bundle represented the last remaining "indirect" container of an actual scalar that is connected. (See "Bundle/Scalar Manipulation" on page 34.)

PortScalars

Destroying a PortScalar first removes it from any PortBundle, detaches the PortScalar from any NetScalars, and then finally destroys the actual PortScalar object.

PortBundles (and PortBusses)

Destroying a PortBundle (or PortBus) first detaches each and every PortScalar directly or indirectly owned by the PortBundle from any attached NetScalar(s) thereby detaching all PortBundles from any NetBundles. Then each Port object (PortBundle, PortBus, or PortScalar) owned by this PortBundle is destroyed. The order of detachment or destruction is implementation-dependent.

Views or Ports used as Describers

Destroying a View or Port that is referenced in another View via the *Describer* relationship of an Inst or PortInst just breaks the *Describer* relationship from the Inst or PortInst. The Inst and PortInst are not themselves destroyed.

Insts (and PortInsts)

Destroying an Inst first detaches each PortInst (on this Inst) from any Net to which it is attached. The Inst is then removed from the list of Insts in the Owner View. Finally the Inst plus all the PortInsts in the PortInsts list (on this Inst) is destroyed.

NOTE: This is the only way the PortInsts may be destroyed.

Persistence

Destroying a persistent container object (a Lib, View or SelectorSet) destroys the object itself and all contained objects, and all persistent data associated with the object. Destroying any other object does not destroy the persistent data associated with that object. The update caused by the destruction must be saved (with LibSave, ViewSave, or SelectorSetSave) in order for the persistent data to be destroyed.

REFERENCE

cfidrLibSave()
cfidrIterNextView()

5.8.3 cfidrObjectFindPropByName

DECLARATION

cfidrPropIdT cfidrObjectFindPropByName(
cfidrObjectIdT object,
cfidrStringT name,
cfidrErrorT *error)

DESCRIPTION

This function returns the Prop with the specified name that is within the Props relationship defined for the Object specified by object.

RETURN VALUE

The return value is a cfidrPropIdT referring to the Prop just found. If an error occurs, a Null OID is returned.

PARAMETERS

object (input) The OID of the Object from which the Prop is to be found.

name (input) The name of the Prop.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
object is not a usable OID.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no Prop with the specified name exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

5.8.4 cfidrObjectGetObjectType

DECLARATION

cfidrObjectTypeT cfidrObjectGetObjectType(
cfidrObjectIdT object,
cfidrErrorT *error)

DESCRIPTION

This function returns the *ObjectType* attribute for the object specified by *object.*

RETURN VALUE

The return value is the enumerated type cfidrObjectTypeT which knows about all the objects which can be created. Currently, this includes only those leaf-level subtypes. If an error occurs, CFIDR_UNDEFINED_OBJECTTYPE is returned.

object (input) The OID of the object for which the *ObjectType* attribute is desired. The *ObjectType* attribute exists for the base Object supertype, so all OIDs are valid input arguments, even the Null OID.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
object is not a usable OID.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

Passing in a Null OID will return the CFIDR_UNDEFINED_OBJECTTYPE constant.

REFERENCE

The following functions all return objects that need to be further differentiated via the cfidrObjectGetObjectType() function:

cfidrCellGetViews()
cfidrNetBundleGetNets()
cfidrPortGetOwner()
cfidrPortBundleGetPorts()
cfidrPortInstGetOwner()
cfidrPortInstBundleGetPortInsts()
cfidrPropGetOwner()
cfidrIterNextNet()
cfidrIterNextPort()
cfidrIterNextPortInst()

cfidrIterNextView()

5.8.5 cfidrObjectGetProps

DECLARATION

cfidrPropsIdT cfidrObjectGetProps(
cfidrObjectIdT object,
cfidrErrorT *error)

DESCRIPTION

This function initiates a traversal of all of the Property objects at the end of the *Props* relationship defined for the object specified by *object.*

RETURN VALUE

The return value is a cfidrPropsIdT referring to an Iterator ID that iterates over Property objects. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextProp() function returns a Null OID.

PARAMETERS

object (input) The OID of an object for which the *Props* relationship is to be traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
object is not a usable OID.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Since all objects can have associated Properties (including Properties), this function accepts all usable OIDs.

POST-CONDITIONS

The **cfidrIterNextProp()** function returns the next Property OID in the *Props* relationship.

If no objects are currently present in the *Props* relationship, an error is not returned but the first call to

cfidrIterNextProp() returns a Null OID.

REFERENCE

cfidrIterNextProp()

5.8.6 cfidrObjectIsNull

DECLARATION

cfidrBooleanT cfidrObjectIsNull(
cfidrObjectIdT object,
cfidrErrorT *error)

DESCRIPTION

This function determines whether the OID specified via *object* is a Null OID.

NOTE: This is the only mechanism which should be used to determine whether an OID is the Null OID, since different PI implementations may define the Null OID differently. The equality operator, "==" should not be used to test for Null.

RETURN VALUE

The return value denotes whether or not the object is a Null OID. If it is, CFIDR_TRUE is returned. Otherwise, CFIDR_FALSE is returned.

PARAMETERS

object (input) The OID being compared to the Null OID.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
object is not a usable OID.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Object must be usable or the Null OID.

REFERENCE

cfidrObjectIsUsable()
cfidrPIGetNullId()

5.8.7 cfidrObjectIsSame

DECLARATION

cfidrBooleanT cfidrObjectIsSame(
cfidrObjectIdT object1,
cfidrObjectIdT object2,
cfidrErrorT *error)

DESCRIPTION

This function determines whether the OIDs specified via *object1* and *object2* refer to the same object.

NOTE: This is the only mechanism which should be used to determine whether two OIDs refer to the same object since OIDs must be treated as abstract data types. The equality operator, "==" should not be used to test for OID equality because that may not correspond to objects being the same.

RETURN VALUE

The return value denotes whether or not the objects passed in refer to the same object. If they do, CFIDR_TRUE is returned. Otherwise, CFIDR_FALSE is returned.

PARAMETERS

object1 (input) The OID of the first object being compared.

object2 (input) The OID of the second object being compared.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
either object1 or object2 is not a usable OID.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The OIDs passed as input arguments should refer to usable objects or be the Null OID.

5.8.8 cfidrObjectIsUsable

DECLARATION

cfidrBooleanT cfidrObjectIsUsable(
cfidrObjectIdT object,
cfidrErrorT *error)

DESCRIPTION

This function determines whether the OID specified by *object* references a usable object.

NOTE: This is the only mechanism which should be used to determine whether an OID references a usable object. Since an OID is an abstract data type, the application can assume nothing about whether it represents a usable object.

RETURN VALUE

The return value denotes whether the object is usable or not. If it is, then CFIDR_TRUE is returned. Otherwise, CFIDR_FALSE is returned.

PARAMETERS

object (input) The OID of the object being checked for usability.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The Null OID is not usable.

cfidrLibPurge() and cfidrViewPurge() cause objects to become unusable.

REFERENCE

cfidrObjectDestroy()
cfidrLibPurge()
cfidrViewPurge

5.9 PI Environment Functions

5.9.1 cfidrPICreateLib

DECLARATION

cfidrLibIdT cfidrPICreateLib(
cfidrStringT name,
cfidrErrorT *error)

DESCRIPTION

This function creates a Lib (library) object that is referred to by *name*.

RETURN VALUE

The return value is a cfidrLibIdT which refers to the Lib object just created. If an error occurs, a Null OID is returned.

PARAMETERS

name (input) The string representing the name of the Lib to be created.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_ALREADY_EXISTS:
a Lib with the same name already exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The system implementing the DR-PI is responsible for knowing about all libraries created and saved in previous sessions or created in the current session when determining whether the Lib already exists. The Lib need not be opened to be considered as existing.

Each system implementing the DR-PI is responsible for creating and maintaining where the Lib's data is

stored. The application using the PI need not have any prior knowledge as to where the data physically resides.

The *name* argument must conform to the character set rules for name strings.

If cfidrPIInit() has not been called before, this function will call it before proceeding.

POST-CONDITIONS

A created Lib is not persistent across session boundaries until it is saved.

REFERENCE

cfidrPIInit()
cfidrLibSave()

5.9.2 cfidrPIFindLibByName

DECLARATION

```
cfidrLibIdT cfidrPIFindLibByName(
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the Lib with the specified name.

RETURN VALUE

The return value is a cfidrLibIdT referring to the Lib just found. If an error occurs, a Null OID is returned.

PARAMETERS

name (input) The name of the Lib.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no Lib with the specified name exists.

CFIDR_INTERNAL_SYSTEM_ERROR:

some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

If the Lib was not opened prior to this function call, it is opened automatically, with mode CFIDR_READ.

5.9.3 cfidrPIGetErrorText

DECLARATION

cfidrStringT cfidrPIGetErrorText(
cfidrErrorT errorCode,
cfidrErrorT *error)

DESCRIPTION

This function returns a pre-defined string describing the nature of the error specified by *errorCode*. All errors returned by PI functions defined in this document will have a pre-defined string.

RETURN VALUE

The value returned by the function is a cfidrStringT containing the pre-defined text. In the case of an error the null string ("") is returned.

PARAMETERS

errorCode (input) The error code for which a textual description is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ERROR_CODE:
errorCode is not in the range of valid error codes.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The memory for the string is managed by the DR-PI. The string's value remains valid until the next

execution of any PI function which has a cfidrStringT return value.

5.9.4 cfidrPIGetLibs

DECLARATION

cfidrLibsIdT cfidrPIGetLibs(
cfidrErrorT *error)

DESCRIPTION

This function initiates a traversal of all the Libraries known by the DR-PI (i.e., no object currently defined has a *Libs* relationship defined for it).

RETURN VALUE

The return value is a cfidrLibsIdT referring to an Iterator ID that iterates over Libraries.

If an error occurs in cfidrPIGetLibs(), the *error* output argument is set and a valid Iterator ID is returned. In this case, calling the cfidrIterNextLib() function returns the Null OID.

PARAMETERS

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INTERNAL_SYSTEM_ERROR:
a system error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The **cfidrIterNextLib()** function returns the OID of the next Lib known by the DR-PI.

If no Libraries are known by the PI, an error is not returned but the first call to **cfidrIterNextLib()** returns the Null OID.

If **cfidrPIInit()** has not been called before, this function will call it before proceeding.

REFERENCE

cfidrIterNextLib()
cfidrPIInit()

5.9.5 cfidrPIGetNullId

DECLARATION

cfidrObjectIdT cfidrPIGetNullId(
cfidrErrorT *error)

DESCRIPTION

This function returns the definition of a Null OID.

RETURN VALUE

The value returned by this function is a cfidrObjectIdT which is the representation of a Null OID for the system implementing the DR-PI. If an error occurs, the return value is undefined and the error return parameter should be set to an appropriate error code.

PARAMETERS

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

If **cfidrPIInit()** has not been called before, this function will call it before proceeding.

REFERENCE

cfidrPIInit()

5.9.6 cfidrPIInit

DECLARATION

cfidrVoidT cfidrPIInit(
cfidrErrorT *error)

DESCRIPTION

This function initializes the programming interface. Calling most other DR-PI functions before calling

this function will result in an error (at least for those functions accepting or returning OIDs).

PARAMETERS

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

All object IDs are invalid before calling this function for the first time. However, the functions cfidrPIFindLibByName(), cfidrPICreateLib(), cfidrPIOpenView(), and cfidrPIOpenSelectorSet() are called with strings, and the functions cfidrPIGetLibs(), cfidrPIGetSelectorSets(), cfidrPIQuit(), and cfidrPIGetNullId() have no OID parameters, so all of these functions will automatically call cfidrPIInit() if it has not been done before.

POST-CONDITIONS

All OIDs created after calling this function remain usable until the object (or its owner) is destroyed using **cfidrObjectDestroy()**, is purged using cfidrLibPurge(), cfidrViewPurge(), cfidrSelectorSetPurge() or a call to **cfidrPIQuit()** is made.

There are no side effects if the function is called multiple times without an ending **cfidrPIQuit()**.

REFERENCE

cfidrObjectDestroy()
cfidrObjectPurge()
cfidrPIQuit()
cfidrPICreateLib()
cfidrPIOpenLib()
cfidrPIOpenView()
cfidrPIGetLibs()
cfidrPIGetNullId()

5.9.7 cfidrPIOpenLib

DECLARATION

```
cfidrLibIdT cfidrPIOpenLib(
cfidrStringT name,
cfidrAccessModeT mode,
```

cfidrErrorT *error)

DESCRIPTION

This function opens up a Lib (library) specified via *name*. Mode can be CFIDR_READ or CFIDR_UPDATE. A Lib must be opened for update for Cells, Views, and Properties to be added to or deleted from the Lib. This function can be used for upgrading a Lib originally opened for CFIDR_READ (possibly via an implicit open) to CFIDR_UPDATE.

RETURN VALUE

The return value is a cfidrLibIdT referring to the Lib just opened. If an error occurs, a Null OID is returned.

PARAMETERS

name (input) A string representing the name of the Lib to be opened.

mode (input) The mode to open the Lib - CFIDR_READ for read-only access, CFIDR_UPDATE to allow modification.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_LIB_ALREADY_OPEN:
a Lib with the specified name is already open.

CFIDR_OBJECT_NOT_FOUND:
no Lib with the specified name exists.

CFIDR_LIB_OPEN_FAILED:
a Lib with the specified name cannot be opened for some reason. For instance, the user may have no access rights.

If the request is for CFIDR_READ, this error means that either the user has no read access rights.

If the request is for CFIDR_UPDATE, this error means that either the user has no write access rights.

CFIDR_INVALID_ACCESSMODE:
mode is outside of the range of legal access mode values.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:

no error occurred.

PRE-CONDITIONS

If the Lib had been created and saved in a previous session, then it is by definition available to be opened in the current session.

If the Lib does not exist, it is not created automatically.

POST-CONDITIONS

Opening a Lib means the object hierarchy owned by the Lib is made available until Views are encountered. Views are opened independently via **cfidrPIOpenView()** or by traversing the iterator returned for a Cell's *Views* attribute (**cfidrCellGetViews()**) using the **cfidrIterNextView()** function.

If the Lib has been previously opened but not purged in the current session, then opening it again returns the same OID. In this case, the *error* output argument should be set to CFIDR_LIB_ALREADY_OPEN to denote this fact even though it is not an error.

If the library was already open with CFIDR_READ but the mode is now CFIDR_UPDATE, the changes will now be allowed.

If the library was already open with CFIDR_UPDATE but the mode is now CFIDR_READ, then no further changes will be allowed, but any prior changes (made after the open for update but before the re-open for read-only) will be made persistent if a cfidrLibSave () is done.

If **cfidrPIInit()** has not been called before, this function will call it before proceeding.

REFERENCE

cfidrCellGetViews()
cfidrPIOpenView()
cfidrIterNextView()
cfidrPIInit()

5.9.8 cfidrPIOpenView

DECLARATION

```
cfidrViewIdT cfidrPIOpenView(
cfidrStringT libName,
cfidrStringT cellName,
cfidrStringT viewName,
cfidrAccessModeT mode,
cfidrErrorT *error)
```

DESCRIPTION

This function opens the View specified by *libName, cellName,* and *viewName.* Mode can be

CFIDR_READ or CFIDR_UPDATE. A View must be opened for update for objects to be added. This function can be used for upgrading a View originally opened for CFIDR_READ (possibly via an implicit open) to CFIDR_UPDATE.

RETURN VALUE

The return value is a cfidrViewIdT referring to the View just or already opened. If an error occurs, a Null OID is returned.

PARAMETERS

libName (input) A string representing the name of the Lib in which the View is supposed to exist.

cellName (input) A string representing the name of the Cell in which the View is supposed to exist.

viewName (input) A string representing the name of the View to be opened.

mode (input) The mode to open the View - CFIDR_READ for read-only access, CFIDR_UPDATE to allow modification.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_NAME:
one or more of the libName cellName viewName parameters is not a valid string or is not a legal name.

CFIDR_VIEW_ALREADY_OPEN:
a View with the specified libName cellName viewName is already open.

CFIDR_OBJECT_NOT_FOUND:
no View with the specified libName cellName viewName exists.

CFIDR_VIEW_OPEN_FAILED:
a View with the specified name cannot be opened for some reason. For instance, the user may have no access rights.

If the request is for CFIDR_READ, this error means that either the user has no read access rights.

If the request is for CFIDR_UPDATE, this error means that either the user has no write access rights.

CFIDR_INVALID_ACCESSMODE:
mode is outside of the range of legal access mode values.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The View is not created automatically if it does not exist already. See **cfidrCellCreateEncapsulatedView()** and **cfidrCellCreateNetlistView()** for details on how to create new views.

POST-CONDITIONS

Opening a NetlistView means the object hierarchy owned by the NetlistView is made available.

Opening an EncapsulatedView means that its ViewType and *Key* attributes are now available. An implementor of this function may optionally also open the encapsulated information referenced by the EncapsulatedView---if that is meaningful. However, this is not required since the particulars of encapsulated information are, by definition, system dependent.

If the View has been previously opened but not purged in the current session, then opening it again returns the same OID. In this case, the *error* output argument should be set to CFIDR_VIEW_ALREADY_OPEN to denote this fact even though it is not an error.

If the Lib owning the Cell and View has not been opened already, it will be opened with the same Mode parameter. If that Lib open fails, the error return from this function will be the same as the error return from a call to **cfidrPIOpenLib()** had been called with that Mode. If the View open fails, the open status of the Lib is indeterminate.

If the View was already open with CFIDR_READ but the mode is now CFIDR_UPDATE, then changes will now be allowed.

If the View was already open with CFIDR_UPDATE but the mode is now CFIDR_READ, then no further changes will be allowed, but any prior changes (made after the open for update but before the re-open for read-only) will be made persistent if a cfidrLibSave() or cfidrViewSave() is done.

If **cfidrPIInit()** has not been called before, this function will call it before proceeding.

Opening a particular NetlistView has two non-obvious side effects:

(1) if an Inst has that NetlistView in its *Describer* attribute, it will not be necessary to automatically open that NetlistView when accessing *Describer*,

(2) for any Inst whose *Owner* attribute is that particular NetlistView, it will be possible to access the *Describer* attribute without an (automatic) open if the NetlistView in *Describer* is open already.

REFERENCE

cfidrPIInit()
cfidrPIOpenLib()
cfidrCellCreateEncapsulatedView()
cfidrCellCreateNetlistView()

5.9.9 cfidrPIQuit

DECLARATION

cfidrVoidT cfidrPIQuit(
cfidrErrorT *error)

DESCRIPTION

This function terminates the execution of the programming interface. Calling any other DR-PI function after calling this function will normally result in an error (at least for those functions accepting or returning OIDs).

PARAMETERS

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

cfidrPIQuit() may have already been called. This is not an error.

POST-CONDITIONS

All OIDs created since calling **cfidrPIInit()** are made unusable after calling this function. Subsequently calling **cfidrPIInit()** starts another session boundary and does not subsequently make usable any OIDs defined prior to the call to **cfidrPIQuit()**.

Quitting the DR-PI does not automatically save any information. The application must call **cfidrLibSave()** or cfidrViewSave() in order to update the persistent data.

REFERENCE

cfidrPIInit()
cfidrLibSave()
cfidrViewSave()

Table of Contents Next Chapter

Table of Contents Previous Chapter

5.10 Port Functions

5.10.1 cfidrPortGetOwner

DECLARATION

cfidrObjectIdT cfidrPortGetOwner(
cfidrPortIdT port,
cfidrErrorT *error)

DESCRIPTION

This function returns the object (View or PortBundle) at the end of the *Owner* relationship for the Port specified by *port*.

RETURN VALUE

The return value is a cfidrObjectIdT referring to the View or PortBundle at the end of the *Owner* relationship. If an error occurs, a Null OID is returned.

PARAMETERS

port (input) The OID of the Port whose *Owner* relationship is to be followed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
port is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
port is not the OID of a Port

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *Owner* relationship is defined when an object is created but can be modified later by the **cfidrPortSetOwnerView()** and **cfidrPortSetOwnerPortBundle()** functions.

POST-CONDITIONS

Use the cfidrObjectGetObjectType() function to determine the Owner's object type.

REFERENCE

cfidrObjectGetObjectType()
cfidrPortSetOwnerPortBundle()
cfidrPortSetOwnerView()

5.10.2 cfidrPortGetPosition

cfidrInt32T cfidrPortGetPosition(
cfidrPortIdT port,
cfidrErrorT *error)

DESCRIPTION

This function returns the position of the specified *port* in its owner.

RETURN VALUE

The return value is a cfidrInt32T. If an error occurs, a -1 is returned.

PARAMETERS

port (input) The OID of a Port.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
port is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
port is not the OID of a Port.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The owner of the Port may be either the View or a PortBundle (or PortBus).

POST-CONDITIONS

The returned position will be 0 for the first Port in the owner up to size-1 for the last position, where size is the number of ports in the list of ports owned by *port*'s owner.

REFERENCE

cfidrPortSetOwnerPortBundle()
cfidrPortBundleCreatePortBundle()
cfidrNetlistViewCreatePortBundle()

5.10.3 cfidrPortSetOwnerPortBundle

DECLARATION

```
cfidrVoidT cfidrPortSetOwnerPortBundle(
cfidrPortIdT port,
cfidrPortBundleIdT owner,
cfidrInt32T position,
cfidrErrorT *error)
```

DESCRIPTION

This function changes the Owner of the specified Port to the PortBundle specified by *owner* and inserts it at the specified *position*. The port is removed from its previous parent (either PortBundle or View) in the process.

PARAMETERS

port (input) The OID of the Port whose owner is to be changed.

owner (input) The OID of the new owning PortBundle.

position (input) The desired location of the port within its new owner PortBundle. If position is less than 0 or greater than or equal to the number of members in the list of Ports owned by owner, the new Port is appended as the last member of the list. Otherwise, it is inserted at the specified position in the list (0 is the first member).

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
port is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
port is not the OID of a Port.

CFIDR_UNUSABLE_PORTBUNDLE_OID:
owner is not a usable OID.

CFIDR_INVALID_PORTBUNDLE_TYPE:
owner is not the OID of a PortBundle

CFIDR_NAME_IN_USE:
the name of the Port matches the name of a Port already owned by the PortBundle.

NOTE: This code is not returned when only the position is being changed but not the owner.

CFIDR_CROSS_VIEW_OPERATION:
the owner PortBundle and the port member belong to different Views.

CFIDR_MEMBER_RECURSION:
the owner and port OIDs refer to the same object.

CFI_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Both the Port and PortBundle must belong to the same parent View.

The name of the Port must be unique with respect to all other Ports owned by the owner PortBundle.

It is not permitted to add a Port as a member of itself.

The *port* may already be a member of the specified *owner* PortBundle. In this case only the *position* is changed (if it is different from the previous *Position*). Do not return the code CFIDR_NAME_IN_USE in this case.

POST-CONDITIONS

If port is already a member of a PortBundle, it will be removed from the list of members of its current owner. All remaining members of that list will be moved down one position.

If port is a member of a NetlistView, it will be removed from the list of members of that NetlistView. All remaining members of that list will be moved down one position.

The Port is added to the list of Ports owned by owner at position. If position = 0, the new Port will be added as the first member of the list. If position is less than 0 or greater than or equal to the number of members in the list, the new Port is appended to the end of the list. Otherwise, the Port is inserted at the specified position (0 based). The position of all members whose positions were greater than or equal to

position, will have their position incremented by 1.

For example, presume a View has Port members:
View Ports = {A, B, C0, 1, 2}

and PortBundle B has members:
B's Port members = {b1, b2, b3}

Setting the owner of Port C0 to position 1 of B results in:
View ports = {A, B, 1, 2}
B's port members = {b1, C0, b2, b3}.

Position specifies the position of the Port *AFTER* the function executes, even in the case where Owner specifies the current owner of Port.

If the Port is connected to a Net, then an implicit detach of the parent bundles will result if that was the last such connection, and an implicit attach of the new parent bundles will result if it is the first such connection.

See the discussion, "NetBundle Connection to Port[Inst]Bundle is a Derived Relationship" on page 4-35, for additional implications.

REFERENCE

cfidrPortSetOwnerView()
cfidrPortBundleGetPorts()
cfidrPortBundleCreatePortBundle()
cfidrNetlistViewCreatePortBundle()

5.10.4 cfidrPortSetOwnerView

DECLARATION

```
cfidrVoidT cfidrPortSetOwnerView(
cfidrPortIdT port,
cfidrInt32T position,
cfidrErrorT *error)
```

DESCRIPTION

This function changes the *Owner* of a Port owned by a PortBundle to the NetlistView which was its ultimate, indirect owner. If the Port is already owned by a NetlistView the *Owner* is not changed. The Port specified by *port* is then moved (if necessary) such that its final position in the Ports of the NetlistView is position.

PARAMETERS

port (input) The OID of the Port whose *Owner* is to be changed.

position (input) The desired location of the port within the NetlistView. If position is less than 0 or greater than or equal to the number of members in the list of Ports owned by owner, the new Port is appended as the last member of the list. Otherwise, it is inserted at the specified position in the list (0 is the first member).

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
port is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
port is not the OID of a Port (PortBundle, PortBus, or PortScalar).

CFIDR_NAME_IN_USE:
the name of the Port matches the name of a Port already owned by the NetlistView.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

If the Port is owned by a PortBundle, the name of the Port must be unique with respect to all other Ports owned by the NetlistView which is the ultimate owner of the PortBundle.

The Port may be owned by a NetlistView. In this case only the *Position* will be changed (if the specified *position* is different than the current *Position*). Do not return the code CFIDR_NAME_IN_USE in this case.

POST-CONDITIONS

If port is owned by a PortBundle, it is removed from that PortBundle.

If port is owned by a PortBundle the ultimate owning NetlistView is determined by getting the *Owner* of the PortBundle. If *Owner* is a PortBundle then *Owner* of that PortBundle is obtained. This is repeated until an *Owner* of type NetlistView is returned.

The positions of all Ports within its previous Owner that are greater than or equal to the port's original *Position* will be decremented by one.

The Port is then inserted in the list of Ports of the NetlistView at position. If position = 0, the new Port will be added as the first member of the list. If position is less than 0 or greater than or equal to the

number of members in the list, the new Port is appended to the end of the list. Otherwise, the Port is inserted at the specified position (0 based). The position of all members whose positions were greater than or equal to position, will have their position incremented by 1.

For example, presume a View has Port members:
View Ports = {A, B, C0, 1, 2}

and PortBundle B has members:
B's Port members = {b1, **b2**, b3}

Setting the owner of Port **b2** to position -1 of the NetlistView results in:
View ports = {A, B, C0, 1, 2, **b2**}
B's port members = {b1, b3}.

Position specifies the position of the Port *AFTER* the function executes, even in the case where Owner specifies the current owner of Port.

If the Port is connected to a Net, then an implicit detach of the parent bundles will result if that was the last such connection.

See the discussion, "NetBundle Connection to Port[Inst]Bundle is a Derived Relationship" on page 4-35, for additional implications.

REFERENCE

cfidrPortSetOwnerPortBundle()
cfidrPortBundleGetPorts()
cfidrPortBundleCreatePortBundle()
cfidrNetlistViewCreatePortBundle()

5.10.5 cfidrPortBundleCreatePortBundle

DECLARATION

```
cfidrPortBundleIdT cfidrPortBundleCreatePortBundle(
cfidrPortBundleIdT owner,
cfidrStringT name,
cfidrInt32T position,
cfidrErrorT *error)
```

DESCRIPTION

This function creates a PortBundle object owned by the PortBundle specified via *owner*.

RETURN VALUE

The return value is a cfidrPortBundleIdT referring to the PortBundle object just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the PortBundle owning the PortBundle being created.

name (input) The string representing the name of the PortBundle being created.

position (input) The integer specifying the position of the PortBundle in the list of Ports owned by *owner*. A *position* value less than 0 or greater than or equal to the number of ports indicates the new PortBundle is put at end of the list. Any other value, including 0, indicates the actual position in the list.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a PortBundle.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_NAME_IN_USE:
the name parameter specifies a name already in use in the same name scope but not for a PortBundle.

CFIDR_OBJECT_ALREADY_EXISTS:
a PortBundle with the same name already exists for this *owner*.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

A Port named *name* that is owned by the same PortBundle as specified by *owner* cannot already exist.

POST-CONDITIONS

The PortBundle is created with no members initially. Later, Ports or PortBundles may either be created in or have their owner changed to be this PortBundle. Furthermore, this PortBundle may later have its owner changed to the View or to another PortBundle.

The PortBundle is created and added to the list of Ports owned by the PortBundle specified by *owner* at the position specified by position. If position = 0, the new PortBundle will be added as the first member of the list. If position is less than 0 or greater than or equal to the number of members in the list, the new PortBundle is appended to the end of the list. Otherwise, the PortBundle is inserted at the specified position (0 based). The position of all Ports whose positions were greater than or equal to position, will have their position incremented by 1. For example, presume the *owner* initially has Ports:
A, B, C0, 1, 2.

Adding a new member X at position 1 results in:
A, X, B, C0, 1, 2
Note that B had position 1 before and has position 2 after X is created.

Adding X at position 0 causes X to be at the beginning of the list:
X, A, B, C0, 1, 2

Adding X at an invalid position such as -1 (or any value greater than 4) causes creation of the new member at the end of the list:
A, B, C0, 1, 2, X

REFERENCE

cfidrPortBundleGetOwner()
cfidrPortBundleGetPorts()
cfidrPortSetOwnerPortBundle()
cfidrPortSetOwnerView()
cfidrNetlistViewCreatePortBundle()
cfidrNetlistViewCreatePortScalar()
cfidrNetlistViewGetPorts()

5.10.6 cfidrPortBundleCreatePortBus

DECLARATION

cfidrPortBusIdT cfidrPortBundleCreatePortBus(
cfidrPortBundleIdT owner,
cfidrStringT name,
cfidrInt32T position,
cfidrInt32T start
cfidrInt32T step
cfidrErrorT *error)

DESCRIPTION

This function creates a PortBus object located at the position *position* in the list of Ports owned by the PortBundle specified via *owner*. The PortBus *Name* attribute is set to *name*. The PortBus *Start* attribute is set to *start*. The PortBus *Step* attribute is set to *step*.

RETURN VALUE

The return value is a cfidrPortBusIdT referring to the PortBus object just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the PortBundle owning the PortBus being created.

name (input) The string representing the name of the PortBus being created.

position (input) The integer specifying the position of the PortBus in the list of Ports owned by *owner*. A *position* value less than 0 or greater than or equal to the number of ports indicates the new PortBus is put at end of the list. Any other value, including 0, indicates the actual position in the list.

start (input) The Int32 defining the index of position 0 in the bus.

step (input) The Int32 defining the difference in indexes of positions n and n+1 in the bus. This number must be non-zero.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a PortBundle.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_NAME_IN_USE:
the name parameter specifies a name already in use in the same name scope but not for a PortBus.

CFIDR_OBJECT_ALREADY_EXISTS:
a PortBus with the same name already exists.

CFIDR_INVALID_VALUE
Step is 0.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

A Port (PortScalar, PortBundle, or PortBus) with the given *name* cannot already exist in the PortBundle specified by *owner*.

POST-CONDITIONS

The newly created PortBus will initially have no members and a size of 0.

The PortBus created by this function can later be changed to have a different PortBundle as its owner using the **cfidrPortSetOwnerPortBundle()** function or to have a View as its owner using the **cfidrPortBundleSetOwnerView()** function.

The Start and Step attributes may be changed later by use of the **cfidrPortBusSetStart()** and **cfidrPortBusSetStep()** functions.

REFERENCE

cfidrPortBundleGetSize()
cfidrPortBundleGetNets()
cfidrPortBundleGetPorts()
cfidrPortBundleSetOwnerView()
cfidrPortBusGetStart()
cfidrPortBusGetStep()
cfidrPortBusSetStart()
cfidrPortBusSetStep()
cfidrPortSetOwnerPortBundle()
cfidrNetlistViewGetPorts()

5.10.7 cfidrPortBundleCreatePortScalar

DECLARATION

```
cfidrPortScalarIdT cfidrPortBundleCreatePortScalar(
cfidrPortBundleIdT owner,
cfidrStringT name,
cfidrInt32T position,
cfidrPortDirectionT direction,
cfidrErrorT *error)
```

DESCRIPTION

This function creates a PortScalar object owned by the PortBundle specified via *owner* and located at *position* in the list of Ports owned by *owner*. The PortScalar name attribute is set to *name*. The PortScalar direction attribute is set to *direction*.

RATIONALE

The PortScalar is created with all required attributes set including the derived attribute of position.

RETURN VALUE

The return value is a cfidrPortScalarIdT referring to the PortScalar just created. If an error occurs a Null OID is returned.

PARAMETERS

owner (input) The OID of the PortBundle owning the PortScalar to be created.

name (input) The string representing the name of the PortScalar.

position (input) The integer specifying the position of the PortScalar in the list of Ports owned by *owner*. A *position* value less than 0 or greater than or equal to the number of ports indicates the new Port is put at end of the list. Any other value, including 0, indicates the actual position in the list.

direction (input) The *Direction* attribute to assign to the created PortScalar.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a PortBundle.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_NAME_IN_USE:
the name parameter specifies a name already in use in the same name scope but not for a PortScalar.

CFIDR_OBJECT_ALREADY_EXISTS:
a PortScalar with the same *name* already exists for *owner*.

CFIDR_INVALID_DIRECTION:
direction is not a legal PortDirection.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

A Port (Scalar, Bundle, or Bus) with the given *name* cannot already be owned by the PortBundle specified by *owner*. However, a Port with the same name may exist indirectly in the View or in another PortBundle. This is allowed since the scope of Port names is the owning View or the owning PortBundle.

POST-CONDITIONS

The *owner* of the PortScalar created by this function can later be changed to the View or to a PortBundle provided no other Port has the same *name* in the new owner.

The PortScalar is created and added to the list of Ports owned by the NetlistView specified by *owner* at the position specified by position. If position = 0, the new PortScalar will be added as the first member of the list. If position is less than 0 or greater than or equal to the number of members in the list, the new PortScalar is appended to the end of the list. Otherwise, the PortScalar is inserted at the specified position (0 based). The position of all Ports whose positions were greater than or equal to position, will have their position incremented by 1. For example, presume a View has Ports:
A, B, C0, 1, 2.

Adding a new member X at position 1 results in:
A, X, B, C0, 1, 2
Note that B had position 1 before and has position 2 after X is created.

Adding X at position 0 causes X to be at the beginning of the list:
X, A, B, C0, 1, 2

Adding X at an invalid position such as -1 (or any value greater than 4) causes creation of the new member at the end of the list:
A, B, C0, 1, 2, X

REFERENCE

cfidrPortBundleCreatePortBundle()
cridrPortBundleCreatePortScalar()
cfidrPortBundleGetOwner()
cfidrPortBundleGetPorts()
cfidrPortSetOwnerPortBundle()
cfidrPortSetOwnerView()
cfidrNetlistViewGetPorts()

5.10.8 cfidrPortBundleFindPortByName

DECLARATION

```
cfidrPortIdT cfidrPortBundleFindPortByName(
cfidrPortBundleIdT portBundle,
```

cfidrStringT name,
cfidrErrorT *error)

DESCRIPTION

This function returns the Port with the specified name that is within the Ports relationship defined for the PortBundle specified by portBundle.

RETURN VALUE

The return value is a cfidrPortIdT referring to the Port just found. If an error occurs, a Null OID is returned.

PARAMETERS

portBundle (input) The OID of the PortBundle from which the Port is to be found.

name (input) The name of the Port.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portBundle is not the OID of a PortBundle.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no Port with the specified name exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Only the immediate list of Ports directly owned by the specified *portBundle* is searched by this function. Thus a Port that is indirectly owned by this *portBundle* will not be returned by this function.

5.10.9 cfidrPortBundleFindPortByPosition

cfidrPortIdT cfidrPortBundleFindPortByPosition(
cfidrPortBundleIdT portBundle,
cfidrInt32T position
cfidrErrorT *error)

DESCRIPTION

This function returns the Port at the specified *position* within the specified *portBundle*. The "leftmost" or first position is numbered 0 and the "rightmost" or last position is numbered one less than the size of the bundle.

RETURN VALUE

The return value is a cfidrPortIdT referring to the Port just found. If an error occurs, a Null OID is returned.

PARAMETERS

portBundle (input) The OID of the PortBundle being accessed.

position (input) The position of the returned Port in the PortBundle. A value of 0 will return the Port at the first position. A value one less than the size of the bundle will return the Port at the last position. A value greater than or equal to the size of the bundle or any negative value will cause an error.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portBundle is not the OID of a PortBundle.

CFIDR_INVALID_POSITION:
the *position* parameter is not in the range 0 to one less than the size of the bundle.

NOTE: If the bundle size is 0, this code is returned since no position is valid.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrPortSetOwnerPortBundle()
cfidrPortBundleCreatePort()

5.10.10 cfidrPortBundleGetNetBundles

DECLARATION

cfidrNetBundlesIdT cfidrPortBundleGetNetBundles(
cfidrPortBundleIdT portBundle,
cfidrErrorT *error)

DESCRIPTION

This function initiates a traversal of all of the NetBundles at the end of the *NetBundles* relationship defined for the PortBundle specified by *portBundle*. These are the NetBundles to which this PortBundle connects.

RETURN VALUE

The return value is a cfidrNetBundlesIdT referring to an Iterator ID that iterates over NetBundles. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextNetBundle() function returns a Null OID.

PARAMETERS

portBundle (input) The OID of a PortBundle for which the *NetBundles* relationship is to be traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portBundle is not the OID of a PortBundle.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Returns all the NetBundles which are connected to the PortBundle passed in via *portBundle*. Essentially, this returns the NetBundles which directly or indirectly contain any NetScalars connected to PortScalars owned directly or indirectly by the PortBundle.

POST-CONDITIONS

The **cfidrIterNextNetBundle()** function returns the next NetBundle OID in the NetBundles relationship.

If no objects are currently present in the NetBundles relationship, an error is not returned but the first call to **cfidrIterNextNetBundle()** returns a Null OID.

REFERENCE

cfidrIterNextNet()
cfidrIterNextNetBundle()

5.10.11 cfidrPortBundleGetPorts

DECLARATION

```
cfidrPortsIdT cfidrPortBundleGetPorts(
cfidrPortBundleIdT portBundle,
cfidrIterModeT mode,
cfidrErrorT *error)
```

DESCRIPTION

This function initiates a traversal of all of the Ports (PortBundles, PortBusses, or PortScalars) at the end of the *Ports* relationship defined for the PortBundle specified by *portBundle.*

RETURN VALUE

The return value is a cfidrPortsIdT referring to an Iterator ID that iterates over Ports. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextPort() function returns a Null OID.

PARAMETERS

portBundle (input) The OID of a PortBundle for which the Ports relationship is to be traversed.

mode (input) The mode that determines which Ports are returned. The current valid values are CFIDR_ITER_SCALARS, CFIDR_ITER_BUNDLES, CFIDR_ITER_TOP, and CFIDR_ITER_ALL. See the Pre-Conditions below for more detail on how these modes behave.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portBundle is not the OID of a PortBundle.

CFIDR_INVALID_ITERMODE:
mode is not a valid IterMode.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Ports directly in a PortBundle each have a unique *Position* in the list of *Ports* owned by the PortBundle. The positions in this list are numbered from 0 for the "leftmost" or first position monotonically increasing by 1 up to one less than the size of the list. The order in which the Ports are returned from any given PortBundle is by increasing position starting with 0. The *Position* of a given Port at a given time is determined by the various operations that have been performed on the list of *Ports* including:
(1) the *position* specified when a Port is created in the PortBundle,
(2) the *position* specified when the Port had its *Owner* set to this PortBundle,
(3) appearances and disappearances of Ports at or to the "left" of the *Position* of the given Port due to changes in those Port's *Owner*.

This function can also be used for PortBusses since they are a subtype of PortBundle.

POST-CONDITIONS

The returned list of Ports is accessed by using calls to **cfidrIterNextPort()** on the iterator returned by this function. Each iteration returns another Port OID that is directly or indirectly in the *Ports* relationship based on

(1) the value of the *mode* parameter and
(2) the *Position* of that Port in its *Owner's Ports* relationship and (other than for *mode* =CFIDR_ITER_TOP) any intervening levels of PortBundles.

If no objects are currently present in the *Ports* relationship under the specified *mode*, an error is not returned but the first call to **cfidrIterNextPort()** returns a Null OID.

The Ports that are included when iterating over the **cfidrPortsIdT** iterator returned from this function depends on the *mode* argument specified.

mode=**CFIDR_ITER_TOP**
Returns all those Ports (PortBundles, PortBusses, or PortScalars) which are directly owned by the PortBundle.

mode=**CFIDR_ITER_SCALARS**
Returns the PortScalars directly owned by *portBundle* and recursively owned by PortBundles owned by *portBundle*. No PortBundles are output with this mode, only PortScalars. The order is based on a depth-first recursion.

mode=**CFIDR_ITER_BUNDLES**
Returns all the PortBundles (including PortBusses) owned directly by *portBundle* and recursively by those PortBundles (and PortBusses). No PortScalars are output. The order is determined by depth-first recursion.

mode=**CFIDR_ITER_ALL**
Returns all the Ports (PortBundles, PortBusses, or PortScalars) directly or indirectly owned by *portBundle*. This mode combines the PortScalars returned when *mode*=CFIDR_ITER_SCALARS with the PortBundles returned when *mode*=CFIDR_ITER_BUNDLES.

For all values of *mode* except *mode*=CFIDR_ITER_TOP, note that the Port *Name* scoping rules state that Port *Names* need only be unique within the *Ports* of their *Owner*, which is either the View or a PortBundle. Thus, a Port may exist in the View while others with the same name exist in different PortBundles. Thus, in the Ports returned via these modes, the same name may appear more than once even though they are different Ports and have different OIDs. The function **cfidrObjectIsSame()** must be used in this case to determine if two Ports with the same name are actually the same object.

REFERENCE

cfidrIterNextPort()
cfidrObjectIsSame()
cfidrPortSetOwnerPortBundle()
cfidrPortSetOwnerView()

5.10.12 cfidrPortBundleGetSize

DECLARATION

cfidrInt32T cfidrPortBundleGetSize(
cfidrPortBundleIdT portBundle,
cfidrErrorT *error)

DESCRIPTION

This function returns the number of Ports currently in the specified *portBundle*. This is only the number at the next level and not a count of the total number of scalars in the bundle.

RETURN VALUE

The return value is a cfidrInt32T. If an error occurs, a -1 is returned.

PARAMETERS

portBundle (input) The OID of a PortBundle.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portBundle is not the OID of a PortBundle.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

PortBundles are created initially with Size zero. PortBundle members are created using **cfidrPortBundleCreatePortBundle()** and **cfidrPortBundleCreatePortScalar()**. PortBundle members are added and deleted via **cfidrPortSetOwnerPortBundle()** and **cfidrPortSetOwnerView()**.

REFERENCE

cfidrPortSetOwnerPortBundle()
cfidrPortSetOwnerView()
cfidrPortBundleCreatePortBundle()
cfidrPortBundleCreatePortScalar()
cfidrNetlistViewCreatePortBundle()

5.10.13 cfidrPortBusGetStart

DECLARATION

cfidrInt32T cfidrportBusGetStart(
cfidrportBusIdT portBus,
cfidrErrorT *error)

DESCRIPTION

This function returns the *start* attribute of the specified *portBus*. This specifies the first index value for this bus.

RETURN VALUE

The return value is a cfidrInt32T. If an error occurs, a 0 is returned.

PARAMETERS

portBus (input) The OID of a PortBus.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portBus is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portBus is not the OID of a PortBus.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The start attribute of a PortBus is set at creation time or changed using **cfidrPortBusSetStart().**

REFERENCE

cfidrPortBundleCreatePortBus()
cfidrPortBusSetStart()
cfidrNetlistViewCreatePortBus()

5.10.14 cfidrPortBusGetStep

DECLARATION

```
cfidrInt32T cfidrPortBusGetStep(
cfidrPortBusIdT portBus,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the *step* attribute of the specified *portBus*. This specifies the numeric increment between positions for the index.

RETURN VALUE

The return value is a cfidrInt32T. If an error occurs, a 0 is returned.

PARAMETERS

portBus (input) The OID of a PortBus.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portBus is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portBus is not the OID of a PortBus.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The Step attribute of a PortBus is set at creation time or changed using **cfidrPortBusSetStep()**.

REFERENCE

cfidrPortBundleCreatePortBus()
cfidrPortBusSetStep()
cfidrNetlistViewCreatePortBus()

5.10.15 cfidrPortBusSetStart

DECLARATION

```
cfidrVoidT cfidrPortBusSetStart(
cfidrPortBusIdT portBus,
cfidrInt32T start,
cfidrErrorT *error)
```

DESCRIPTION

This function sets the *start* attribute of the specified *portBus*. This specifies the first index value for this bus.

PARAMETERS

portBus (input) The OID of a PortBus.

start (input) The 32 bit integer number for the first (leftmost) index of the bus.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portBus is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portBus is not the OID of a PortBus.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The Start attribute of the PortBus may be accessed using the function **cfidrPortBusGetStart()**.

REFERENCE

cfidrPortBundleCreatePortBus()
cfidrPortBusGetStart()
cfidrNetlistViewCreatePortBus()

5.10.16 cfidrPortBusSetStep

DECLARATION

```
cfidrVoidT cfidrPortBusSetStep(
cfidrPortBusIdT portBus,
cfidrInt32T step,
cfidrErrorT *error)
```

DESCRIPTION

This function sets the *step* attribute of the specified *portBus*. This specifies the numeric increment between positions for the index.

PARAMETERS

portBus (input) The OID of a PortBus.

step (input) The 32 bit integer number for the step. This number must be non-zero.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portBus is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portBus is not the OID of a PortBus.

CFIDR_INVALID_VALUE:
step is zero.

CFIDR_READ_ONLY:
the containing View has not been opened for update.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The Step attribute of the PortBus may be accessed using the function **cfidrPortBusGetStep()**.

REFERENCE

cfidrPortBundleCreatePortBus()
cfidrPortBusGetStep()
cfidrNetlistViewCreatePortBus()

5.10.17 cfidrPortScalarGetDirection

DECLARATION

cfidrPortDirectionT cfidrPortScalarGetDirection(
cfidrPortScalarIdT portScalar,
cfidrErrorT *error)

DESCRIPTION

This function returns the *Direction* attribute of the PortScalar specified by *portScalar*.

RETURN VALUE

The return value is a cfidrPortDirectionT enumerated type representing the value of the *Direction* attribute.

On error the function returns CFIDR_UNDEFINED_PORTDIRECTION.

PARAMETERS

portScalar (input) The OID of the PortScalar whose Direction attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating

the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portScalar is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portScalar is not the OID of a PortScalar.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

5.10.18 cfidrPortScalarGetNetScalar

DECLARATION

cfidrNetScalarIdT cfidrPortScalarGetNetScalar(
cfidrPortScalarIdT portScalar,
cfidrErrorT *error)

DESCRIPTION

This function retrieves the NetScalar at the end of the *NetScalar* relationship defined for the PortScalar specified by *portScalar*.

RETURN VALUE

The return value is a cfidrNetScalarIdT referring to the NetScalar at the end of the *NetScalar* relationship. If an error occurs or if there is no NetScalar attached to this PortScalar, a Null OID is returned.

PARAMETERS

portScalar (input) The OID of the PortScalar containing the *NetScalar* relationship which is being traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portScalar is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:

portScalar is not the OID of a PortScalar

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrNetGetPorts()

5.11 PortInst Functions

5.11.1 cfidrPortInstBundleFindPortInstByDescriberName

DECLARATION

```
cfidrPortInstIdT
cfidrPortInstBundleFindPortInstByDescriberName(
cfidrPortInstBundleIdT portInstBundle,
cfidrStringT describerName,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the PortInst with the specified describerName that is within the PortInsts relationship defined for the PortInstBundle specified by portInstBundle.

RETURN VALUE

The return value is a cfidrPortInstIdT referring to the PortInst just found. If an error occurs, a Null OID is returned.

PARAMETERS

portInstBundle (input) The OID of the PortInstBundle from which the PortInst is to be found.

describerName (input) The *DescriberName* of the PortInst.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInstBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInstBundle is not the OID of a PortInstBundle.

CFIDR_INVALID_NAME:
the describerName parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no PortInst with the specified describerName exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Only the immediate list of PortInsts directly owned by the specified *portInstBundle* is searched by this function. Thus a PortInst that is only indirectly owned by this *portInstBundle* will not be returned by this function

---

5.11.2 cfidrPortInstBundleGetNetBundles

DECLARATION

cfidrNetBundlesIdT cfidrPortInstBundleGetNetBundles(
cfidrPortInstBundleIdT portInstBundle,
cfidrErrorT *error)

DESCRIPTION

This function initiates a traversal of all of the NetBundles at the end of the *NetBundles* relationship defined for the PortInstBundle specified by *portInstBundle.* These are the NetBundles to which this PortInstBundle connects.

RETURN VALUE

The return value is a cfidrNetBundlesIdT referring to an Iterator ID that iterates over NetBundles. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextNetBundle() function returns a Null OID.

PARAMETERS

portInstBundle (input) The OID of a PortInstBundle for which the *NetBundles* relationship is to be traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInstBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInstBundle is not the OID of a PortInstBundle.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Returns all the NetBundles which are connected to the PortInstBundle passed in via *portInstBundle*. Essentially, this returns the NetBundles which directly or indirectly contain any NetScalars connected to PortInstScalars directly or indirectly owned by the PostInstBundle.

POST-CONDITIONS

The **cfidrIterNextNetBundle()** function returns the next NetBundle OID in the *NetBundles* relationship.

If no objects are currently present in the *NetBundles* relationship, an error is not returned but the first call to **cfidrIterNextNetBundle()** returns a Null OID.

REFERENCE

cfidrIterNextNet()
cfidrIterNextNetBundle()

5.11.3 cfidrPortInstBundleGetPortInsts

DECLARATION

```
cfidrPortInstsIdT cfidrPortInstBundleGetPortInsts(
cfidrPortInstBundleIdT portInstBundle,
cfidrIterModeT mode,
cfidrErrorT *error)
```

DESCRIPTION

This function initiates a traversal of all of the PortInsts (PortInstBundles, PortInstBusses, or PortInstScalars) at the end of the *PortInsts* relationship defined for the PortInstBundle specified by *portInstBundle*.

RETURN VALUE

The return value is a cfidrPortInstsIdT referring to an Iterator ID that iterates over PortInsts. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextPortInst() function returns a Null OID.

PARAMETERS

portInstBundle (input) The OID of a PortInstBundle for which the *PortInsts* relationship is to be traversed.

mode (input) The mode that determines which PortInsts are returned. The current valid values are CFIDR_ITER_SCALARS, CFIDR_ITER_BUNDLES, CFIDR_ITER_TOP, and CFIDR_ITER_ALL. See the Pre-Conditions below for more detail on how these modes behave.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInstBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInstBundle is not the OID of an PortInstBundle.

CFIDR_INVALID_ITERMODE:
mode is not a valid IterMode.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The PortInsts returned depends on the *mode* argument specified.

mode=**CFIDR_ITER_TOP**
Returns only those PortInsts (PortInstBundles, PortInstBusses, or PortInstScalars) which are owned directly by the PortInstBundle.

mode=**CFIDR_ITER_SCALARS**
Returns all the PortInstScalars owned directly or indirectly by the PortInstBundle. No PortInstBundles are output with this mode.

mode=**CFIDR_ITER_BUNDLES**
Returns all the PortInstBundles owned directly or indirectly by the PortInstBundle. No PortInstScalars are output with this mode.

mode=**CFIDR_ITER_ALL**

Returns all the PortInsts (PortInstBundles, PortInstBusses or PortInstScalars) owned directly or indirectly by the PortInstBundle. This is the union of all the PortInsts returned by the CFIDR_ITER_SCALARS and CFIDR_ITER_BUNDLES modes.

Depth first recursion is used for all modes except CFIDR_ITER_TOP.

POST-CONDITIONS

The **cfidrIterNextPortInst()** function returns the next PortInst OID in the *PortInsts* relationship.

If no objects are currently present in the *PortInsts* relationship, an error is not returned but the first call to **cfidrIterNextPortInst()** returns a Null OID.

The PortInst name scoping rules (inherited from the Port name scoping rules) state that PortInst names are only unique within their immediate owner (either an Inst or a PortInstBundle). Thus, two PortInsts with the same name may exist and be completely different objects. Thus, PortInsts returned via any mode except CFIDR_ITER_TOP may have the same name but in fact be different objects. The **cfidrObjectIsSame()** function can be used to determine if any two of these PortInsts are the same.

REFERENCE

cfidrIterNextPortInst()
cfidrObjectIsSame()
cfidrPortBundleGetPorts()

5.11.4 cfidrPortInstBundleGetSize

DECLARATION

```
cfidrInt32T cfidrPortInstBundleGetSize(
cfidrPortInstBundleIdT portInstBundle,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the number of Ports currently in the specified *portInstBundle*. This is only the number at the next level and not a count of the total number of scalars in the bundle.

RETURN VALUE

The return value is a cfidrInt32T. If an error occurs, a -1 is returned.

PARAMETERS

portInstBundle (input) The OID of a PortInstBundle.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInstBundle is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInstBundle is not the OID of a PortInstBundle.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

PortInstBundles are created only indirectly when Insts are created and the *Describer* NetlistView has PortBundles. The PortInstBundle size is that of the *Describer* PortBundle.

REFERENCE

cfidrPortBundleGetSize()
cridrPortInstGetDescriber()
cfidrNetlistViewCreateInst()

5.11.5 cfidrPortInstBusGetStart

DECLARATION

cfidrInt32T cfidrportInstBusGetStart(
cfidrPortInstBusIdT portInstBus,
cfidrErrorT *error)

DESCRIPTION

This function returns the *start* attribute of the specified *portInstBus*. This specifies the first index value for this bus.

RETURN VALUE

The return value is a cfidrInt32T. If an error occurs, a 0 is returned.

PARAMETERS

portInstBus (input) The OID of a PortInstBus.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInstBus is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInstBus is not the OID of a PortInstBus.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The start attribute of a PortInstBus is that of the start attribute of the *Describer* PortBus.

REFERENCE

cfidrPortInstGetDescriber()
cfidrPortBusGetStart()
cfidrNetlistViewCreateInst()

5.11.6 cfidrPortInstBusGetStep

DECLARATION

cfidrInt32T cfidrPortInstBusGetStep(
cfidrPortInstBusIdT portInstBus,
cfidrErrorT *error)

DESCRIPTION

This function returns the *step* attribute of the specified *portInstBus*. This attribute specifies the numeric increment between positions for the index.

RETURN VALUE

The return value is a cfidrInt32T. If an error occurs, a 0 is returned.

PARAMETERS

portInstBus (input) The OID of a PortInstBus.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInstBus is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInstBus is not the OID of a PortInstBus.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The Step attribute of a PortInstBus is that of the *Describer* PortBus.

REFERENCE

cfidrPortBusGetStep()
cfidrPortInstGetDescriber()
cfidrNetlistViewCreatePortBus()

5.11.7 cfidrPortInstCheckDescriber [2]

DECLARATION

cfidrBooleanT cfidrPortInstCheckDescriber(
cfidrPortInstIdT portInst,
cfidrErrorT *error)

DESCRIPTION

This function checks for discrepancies between PortInst specified via *portInst* and its *Describer* Port.

If portInst is a PortInstBundle or Bus, checking is performed on all PortInsts owned either directly or indirectly by the PortInst (as returned by cfidrPortInstBundleGetPortInsts with mode ITER_ALL).

[2] NOTE: This function can be written entirely using other DR-PI functions.

RETURN VALUE

If no discrepancies are found between the PortInst and the *Describer* Port, CFIDR_FALSE is returned. Otherwise, CFIDR_TRUE is returned and the *error* output argument will denote the discrepancy.

PARAMETERS

portInst (input) The OID of the PortInst to check against its *Describer* Port.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating

the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInst is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInst is not the OID of a PortInst.

CFIDR_DESCRIBER_VIEW_NOT_FOUND:
could not find or access the *Describer* View for the Instance on which portInst resides.

CFIDR_DESCRIBER_PORT_NOT_FOUND:
there is no Port corresponding to portInst on the *Describer* View.

CFIDR_PORT_MEMBERS_DIFFER:
portInst is a bundle and the members differ from the members of the corresponding Describer PortBundle OR the corresponding Describer is a PortScalar. This error is also returned if the PortInst is a PortInstScalar, but the corresponding Describer is a PortBundle. This error is also returned if the owner of the describer of a PortInst is not the same object as the describer of the owner of that PortInst.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

If the *Describer* View is not opened yet, then this function will open it for CFIDR_READ. If the open fails, the error returned from this function will be the error returned from the failed open.

REFERENCE

cfidrNetlistViewCreateInst()

5.11.8 cfidrPortInstGetDescriber

DECLARATION

```
cfidrPortIdT cfidrPortInstGetDescriber(
cfidrPortInstIdT portInst,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the Port at the end of the *Describer* relationship for the PortInst specified by *portInst.*

RETURN VALUE

The return value is a cfidrPortIdT referring to the Port at the end of the *Describer* relationship. If an error occurs, a Null OID is returned.

PARAMETERS

portInst (input) The OID of the PortInst whose *Describer* relationship is to be followed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInst is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInst is not the OID of a PortInst.

CFIDR_DESCRIBER_VIEW_NOT_FOUND:
could not find or access the *Describer* View for the Inst on which portInst resides.

CFIDR_DESCRIBER_PORT_NOT_FOUND:
there is no Port corresponding to portInst on the *Describer* View either because the Describer port does not exist or because it is inconsistent with the PortInst due to scalar versus bundle of differing bundle membership. The function cfidrPortInstCheckDescriber() can be used for more information if this error occurs.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

If the *Describer* View is not opened yet, then this function will open it for CFIDR_READ. If the open fails, the error returned from this function will be the error returned from the failed open.

5.11.9 cfidrPortInstGetDescriberName [2]

DECLARATION

```
cfidrStringT cfidrPortInstGetDescriberName(
cfidrPortInstIdT portInst,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the *DescriberName* (derived) attribute of the PortInst specified by *portInst*.

[2] NOTE: this function can be written entirely using other DR-PI functions.

RETURN VALUE

The return value is a cfidrStringT representing the string value of the *DescriberName* attribute. On error, the function returns "".

PARAMETERS

portInst (input) The OID of the PortInst whose *DescriberName* attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInst is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInst is not the OID of a PortInst.

CFIDR_DESCRIBER_VIEW_NOT_FOUND:
could not find or access the *Describer* View for the Inst on which portInst resides.

CFIDR_DESCRIBER_PORT_NOT_FOUND:
there is no Port corresponding to portInst on the *Describer* View.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

If the *Describer* View is not opened yet, then this function will open it for CFIDR_READ. If the open fails, the error returned from this function will be the error returned from the failed open.

REFERENCE

cfidrNamedObjectGetName()
cfidrPortInstGetDescriber().

5.11.10 cfidrPortInstGetOwner

DECLARATION

cfidrObjectIdT cfidrPortInstGetOwner(
cfidrPortInstIdT portInst,
cfidrErrorT *error)

DESCRIPTION

This function returns the object (Instance or PortInstBundle) at the end of the *Owner* relationship for the PortInst specified by *portInst.* The *Owner* relationship is defined when an object is created and cannot be modified.

RETURN VALUE

The return value is a cfidrObjectIdT referring to the Inst or the PortInstBundle at the end of the *Owner* relationship. If an error occurs, a Null OID is returned.

PARAMETERS

portInst (input) The OID of the PortInst whose *Owner* relationship is to be followed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInst is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInst is not the OID of a PortInst.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

Use the cfidrObjectGetObjectType() function to determine the Owner's object type.

REFERENCE

cfidrObjectGetObjectType()

5.11.11 cfidrPortInstScalarGetDescriberDirection [2]

DECLARATION

cfidrPortDirectionT

cfidrPortInstScalarGetDescriberDirection(
cfidrPortInstScalarIdT portInstScalar,
cfidrErrorT *error)

DESCRIPTION

This function returns the *DescriberDirection* (derived) attribute of the PortInstScalar specified by *portInstScalar*.

[2] NOTE: this function can be written entirely using other DR-PI functions.

RETURN VALUE

The return value is a cfidrPortDirectionT enumerated type representing the value of the *DescriberDirection* attribute.

On error the function returns CFIDR_UNDEFINED_PORTDIRECTION.

PARAMETERS

portInstScalar (input) The OID of the PortInstScalar whose *DescriberDirection* attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInstScalar is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInstScalar is not the OID of a PortInstScalar.

CFIDR_DESCRIBER_VIEW_NOT_FOUND:
could not find or access the *Describer* View for the Inst on which portInstScalar resides.

CFIDR_DESCRIBER_PORT_NOT_FOUND:
there is no PortScalar corresponding to portInstScalar on the *Describer* View.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

If the *Describer* View is not opened yet, then this function will open it for CFIDR_READ. If the open fails, the error returned from this function will be the error returned from the failed open.

REFERENCE

cfidrPortInstGetDescriber().

5.11.12 cfidrPortInstScalarGetNetScalar

DECLARATION

cfidrNetScalarIdT cfidrPortInstScalarGetNetScalar(
cfidrPortInstScalarIdT portInstScalar,
cfidrErrorT *error)

DESCRIPTION

This function retrieves the NetScalar at the end of the *NetScalar* relationship defined for the PortInstScalar specified by *portInstScalar*.

RETURN VALUE

The return value is a cfidrNetScalarIdT referring to the NetScalar at the end of the *NetScalar* relationship. If an error occurs or if there is no NetScalar attached to this PortInstScalar, a Null OID is returned.

PARAMETERS

portInstScalar (input) The OID of the PortInstScalar containing the *NetScalar* relationship which is being traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
portInstScalar is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
portInstScalar is not the OID of a PortInstScalar

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrNetGetPortInsts()
cfidrNetGetPorts()

5.12 Property Functions

5.12.1 cfidrPropGetBooleanValue

DECLARATION

cfidrBooleanT cfidrPropGetBooleanValue(
cfidrPropIdT prop,
cfidrErrorT *error)

DESCRIPTION

This function returns the Boolean *Value* attribute of the Property specified by *prop*.

RETURN VALUE

The return value is either CFIDR_TRUE or CFIDR_FALSE. If an error occurs the function returns CFIDR_FALSE.

PARAMETERS

prop (input) The OID of the Property whose Boolean Value attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
prop is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
prop is not the OID of a Boolean-valued Property.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred

REFERENCE

cfidrPropSetBooleanValue()

5.12.2 cfidrPropGetFloat32Value

DECLARATION

cfidrFloat32T cfidrPropGetFloat32Value(
cfidrPropIdT prop,
cfidrErrorT *error)

DESCRIPTION

This function returns the Float32 *Value* attribute of the Property specified by *prop*.

RETURN VALUE

The return value is a cfidrFloat32T representing a 32-bit floating point number. If an error occurs the function returns 0.0.

PARAMETERS

prop (input) The OID of the Property whose Float32 Value attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
prop is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
prop is not the OID of a Float32-valued Property.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The float returned is guaranteed to be in the range CFIDR_MIN_FLOAT32 to CFIDR_MAX_FLOAT32.

REFERENCE

cfidrPropSetFloat32Value()

5.12.3 cfidrPropGetInt32Value

DECLARATION

cfidrInt32T cfidrPropGetInt32Value(
cfidrPropIdT prop,
cfidrErrorT *error)

DESCRIPTION

This function returns the Int32 *Value* attribute of the Property specified by *prop*.

RETURN VALUE

The return value is a cfidrInt32T representing a 32-bit integer number. If an error occurs the function returns 0.

PARAMETERS

prop (input) The OID of the Property whose Int32 Value attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
prop is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
prop is not the OID of a Int32-valued Property.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The integer returned is guaranteed to be in the range CFIDR_MIN_INT32 to CFIDR_MAX_INT32.

REFERENCE

cfidrPropSetInt32Value()

5.12.4 cfidrPropGetOwner

DECLARATION

cfidrObjectIdT cfidrPropGetOwner(

cfidrPropIdT prop,
cfidrErrorT *error)

DESCRIPTION

This function returns the object at the end of the *Owner* relationship for the Property specified by *prop*. The *Owner* relationship is defined when an object is created and cannot be modified.

RETURN VALUE

The return value is a cfidrObjectIdT referring the object at the end of the *Owner* relationship. Since any object can have properties, the owner can be any object type. If an error occurs, a Null OID is returned.

PARAMETERS

prop (input) The OID of the Prop whose *Owner* relationship is to be followed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
prop is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
prop is not the OID of a Property.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

Use the **cfidrObjectGetObjectType()** function to determine the *Owner*'s object type.

REFERENCE

cfidrObjectCreateProp()
cfidrObjectGetObjectType()

5.12.5 cfidrPropGetStringValue

DECLARATION

cfidrStringT cfidrPropGetStringValue(
cfidrPropIdT prop,

cfidrErrorT *error)

DESCRIPTION

This function returns the String *Value* attribute of the Property specified by *prop*.

RETURN VALUE

The return value is a cfidrStringT representing a string. If an error occurs the function returns "" (the empty string).

PARAMETERS

prop (input) The OID of the Property whose String *Value* attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
prop is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
prop is not the OID of a String-valued Property.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

There are no restrictions on the length or characters used in the string. The memory for the string is managed by the DR-PI. The string's value remains valid until the next execution of any PI function which has a **cfidrStringT** return value.

REFERENCE

cfidrPropSetStringValue()

5.12.6 cfidrPropGetValueType

DECLARATION

cfidrValueTypeT cfidrPropGetValueType(
cfidrPropIdT prop,
cfidrErrorT *error)

DESCRIPTION

This function returns the *ValueType* attribute of the Property specified by *prop*.

RETURN VALUE

The return value is a cfidrValueTypeT enumerated type representing the value of the *ValueType* attribute. If an error occurs, the function returns CFIDR_UNDEFINED_VALUETYPE.

PARAMETERS

prop (input) The OID of the Property whose *ValueType* attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
prop is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
prop is not the OID of a Property.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The value type of a Prop is set by the type of the particular value set for the Prop. See the cfidrPropSet...Value() functions.

REFERENCE

cfidrPropSet...Value() functions.

5.12.7 cfidrPropSetBooleanValue

DECLARATION

```
cfidrVoidT cfidrPropSetBooleanValue(
cfidrPropIdT prop,
cfidrBooleanT value,
cfidrErrorT *error)
```

DESCRIPTION

This function sets the Boolean *Value* attribute for the Property specified via *prop*.

PARAMETERS

prop (input) The OID of the Property to which the *Value* attribute is to be set.

value (input) The desired Boolean value for the *Value* attribute of *prop*.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
prop is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
prop is not the OID of a Property.

CFIDR_READ_ONLY:
the containing object (Lib or View) has not been opened for update.

CFIDR_INVALID_VALUE:
value is not a valid Boolean value.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *value* argument must be either CFIDR_TRUE or CFIDR_FALSE. Any other value will not set the default value.

POST-CONDITIONS

The updated object must be saved in order to be persistent. Purging the object before saving it will result in the loss of the changes.

If the Property's *Value* attribute was other than a Boolean, the setting still takes place and the ValueType is changed to **CFIDR_BOOLEAN**.

REFERENCE

cfidrObjectCreateProp()

5.12.8 cfidrPropSetFloat32Value

DECLARATION

cfidrVoidT cfidrPropSetFloat32Value(
cfidrPropIdT prop,
cfidrFloat32T value,
cfidrErrorT *error)

DESCRIPTION

This function sets the Float32 *Value* attribute for the Property specified via *prop*.

PARAMETERS

prop (input) The OID of the Property to which the *Value* attribute is to be set.

value (input) The 32-bit floating point number to set the *Value* attribute to.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
prop is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
prop is not the OID of a Property.

CFIDR_READ_ONLY:
the containing object (Lib or View) has not been opened for update.

CFIDR_INVALID_VALUE:
value is not a valid Float32 value.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *value* argument must be within the range CFIDR_MIN_FLOAT32 to CFIDR_MAX_FLOAT32.

POST-CONDITIONS

The updated object must be saved in order to be persistent. Purging the object before saving it will result in the loss of the changes.

If the Property's *Value* attribute was other than a Float32, the setting still takes place and the *ValueType*

is changed to **CFIDR_FLOAT32.**

REFERENCE

cfidrObjectCreateProp()

5.12.9 cfidrPropSetInt32Value

DECLARATION

```
cfidrVoidT cfidrPropSetInt32Value(
cfidrPropIdT prop,
cfidrInt32T value,
cfidrErrorT *error)
```

DESCRIPTION

This function sets the Int32 *Value* attribute for the Property specified via *prop.*

PARAMETERS

prop (input) The OID of the Property to which the *Value* attribute is to be set.

value (input) The 32-bit integer number to set the *Value* attribute to.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
prop is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
prop is not the OID of a Property.

CFIDR_READ_ONLY:
the containing object (Lib or View) has not been opened for update.

CFIDR_INVALID_VALUE:
value is not a valid Int32 value.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *value* argument must be within the range CFIDR_MIN_INT32 to CFIDR_MAX_INT32.

POST-CONDITIONS

The updated object must be saved in order to be persistent. Purging the object before saving it will result in the loss of the changes.

If the Property's *Value* attribute was other than a Int32, the setting still takes place and the *ValueType* is changed to CFIDR_INT32.

REFERENCE

cfidrObjectCreateProp()

5.12.10 cfidrPropSetStringValue

DECLARATION

```
cfidrVoidT cfidrPropSetStringValue(
cfidrPropIdT prop,
cfidrStringT value,
cfidrErrorT *error)
```

DESCRIPTION

This function sets the String *Value* attribute for the Property specified via *prop*.

PARAMETERS

prop (input) The OID of the Property to which the *Value* attribute is to be set.

value (input) The string to set the *Value* attribute to. The caller is responsible for managing the memory of this string.

NOTE: DR-PI functions which return **cfidrStringT** values guarantee those strings to not be updated until the next call to a PI function that returns another **cfidrStringT.** These strings can be used as the *value* argument because the **cfidrPropSetStringValue()** function will not affect these strings returned from other PI functions.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
prop is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
prop is not the OID of a Property.

CFIDR_READ_ONLY:
the containing object (Lib or View) has not been opened for update.

CFIDR_INVALID_VALUE:
value is not a valid String.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

No limitations exists as far as the length or character set used in the string defined by the *value* argument.

POST-CONDITIONS

The updated object must be saved in order to be persistent. Purging the object before saving it will result in the loss of the changes.

POST-CONDITIONS If the Property's *Value* attribute was other than a String, the setting still takes place and the *ValueType* is changed to CFIDR_STRING.

REFERENCE

cfidrObjectCreateProp()

5.13 View Functions

5.13.1 cfidrEncapsulatedViewGetKey

DECLARATION

```
cfidrStringT cfidrEncapsulatedViewGetKey(
cfidrEncapsulatedViewIdT encapsulatedView,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the *Key* attribute of the EncapsulatedView specified by *encapsulatedView.*

RETURN VALUE

The return value is a cfidrStringT representing the value of the *Key* attribute. On error the string "CFIDR_UNDEFINED_KEY" is returned.

PARAMETERS

encapsulatedView (input) The OID of the EncapsulatedView whose *Key* attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
encapsulatedView is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
encapsulatedView is not the OID of a EncapsulatedView.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *Key* attribute is set by the **cfidrCellCreateEncapsulatedView()** function.

REFERENCE

cfidrCellCreateEncapsulatedView()

5.13.2 cfidrNetlistViewCreateInst

DECLARATION

```
cfidrInstIdT cfidrNetlistViewCreateInst(
cfidrNetlistViewIdT owner,
cfidrNetlistViewIdT describer,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function creates an Inst object owned by the NetlistView specified via *owner*. The Inst is an instantiation of another NetlistView specified via *describer*. The Inst is referred to by *name*.

RETURN VALUE

The return value is a cfidrInstIdT which refers to the Inst object just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the NetlistView owning the Inst being created.

describer (input) The OID of the NetlistView describing the Inst being created.

name (input) The string representing the *Name* of the Inst to create.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a NetlistView.

CFIDR_READ_ONLY:
the View has not been opened for update.
CFIDR_UNUSABLE_DESCRIBER_OID:
describer is not a usable OID.

CFIDR_INVALID_DESCRIBER_TYPE:
describer is not the OID of a NetlistView.

CFIDR_DESCRIBER_RECURSION:
describer is the same NetlistView object as owner or recursively contains an Instance of owner.

CFIDR_INVALID_NAME:
the *name* parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_ALREADY_EXISTS:
an Inst with the *Name name* already exists.

CFIDR_INTRA_CELL_INSTANTIATION:
The owner of *describer* is the same object as the owner of *owner*.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The NetlistView OID specified by *describer* must refer to a usable NetlistView (i.e., top-down design by creating Insts without a describer NetlistView is disallowed). This also implies that the *describer* NetlistView has been previously created or opened.

The *name* argument must conform to the character set rules for name strings.

An Inst with the given *name* cannot already exist in the NetlistView specified by *owner*.

POST-CONDITIONS

Creating an Inst automatically creates PortInsts (PortInstBundles, PortInstBusses, and PortInstScalars) owned by the Inst. These PortInsts correspond 1-for-1 to Ports in the *describer* NetlistView. They may be retrieved as soon as the Inst is successfully created.

REFERENCE

cfidrNetlistViewGetInsts()

5.13.3 cfidrNetlistViewCreateNetBundle

DECLARATION

```
cfidrNetBundleIdT cfidrNetlistViewCreateNetBundle(
cfidrNetlistViewIdT owner,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function creates a NetBundle object owned by the NetlistView specified via *owner*. The NetBundle is referred to by *name*.

RETURN VALUE

The return value is a cfidrNetBundleIdT referring to the NetBundle object just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the NetlistView owning the NetBundle being created.

name (input) The string representing the *Name* of the NetBundle being created.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a NetlistView.

CFIDR_READ_ONLY:
the owner View has notbeen opened for update.

CFIDR_INVALID_NAME:
the *name* parameter is not a valid string or is not a legal name.

CFIDR_NAME_IN_USE:
the name parameter specifies a name already in use in the same name scope but not for a NetBundle.

CFIDR_OBJECT_ALREADY_EXISTS:
a NetBundle with the same name already exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

A Net (NetScalar, NetBundle, or NetBus) with the given *name* cannot already exist in the NetlistView specified by *owner*.

POST-CONDITIONS

The newly created NetBundle will initially have no members and a size of zero.

The NetBundle created by this function can later be inserted into one or more NetBundle(s).

REFERENCE

cfidrNetBundleGetNets()
cfidrNetBundleInsertNet()
cfidrNetBundleRemoveNet()
cfidrNetlistViewGetNets()

5.13.4 cfidrNetlistViewCreateNetBus

DECLARATION

```
cfidrNetBusIdT cfidrNetlistViewCreateNetBus(
cfidrNetlistViewIdT owner,
cfidrStringT name,
cfidrInt32T start
cfidrInt32T step
cfidrErrorT *error)
```

DESCRIPTION

This function creates a NetBus object owned by the NetlistView specified via *owner*. The NetBus name attribute is set to *name*. The NetBus start attribute is set to *start*. The NetBus step attribute is set to *step*.

RETURN VALUE

The return value is a cfidrNetBusIdT referring to the NetBus object just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the NetlistView owning the NetBundle being created.

name (input) The string representing the name of the NetBundle being created.

start (input) The Int32 defining the index of position 0 in the bus.

step (input) The Int32 defining the difference in indexes of positions n and n+1 in the bus. This number must be non-zero.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a NetlistView.

CFIDR_READ_ONLY:
the owner View has not been opened for update.

CFIDR_INVALID_NAME:
the *name* parameter is not a valid string or is not a legal name.

CFIDR_NAME_IN_USE:
the name parameter specifies a name already in use in the same name scope but not for a NetBus.

CFIDR_OBJECT_ALREADY_EXISTS:
a NetBus with the same name already exists.

CFIDR_INVALID_VALUE
Step is 0.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

A Net (NetScalar, NetBundle, or NetBus) with the given *name* cannot already exist in the NetlistView specified by *owner*.

POST-CONDITIONS

The newly created NetBus will initially have no members and a size of 0.

The NetBus created by this function can later be inserted into one or more NetBundle(s).

The Start and Step attributes may be changed later by use of the **cfidrNetBusSetStart()** and **cfidrNetBusSetStep()** functions.

REFERENCE

cfidrNetBundleGetSize()
cfidrNetBundleGetNets()
cfidrNetBundleInsertNet()
cfidrNetBundleRemoveNet()
cfidrNetBusGetStart()
cfidrNetBusGetStep()
cfidrNetBusSetStart()
cfidrNetBusSetStep()
cfidrNetlistViewGetNets()

5.13.5 cfidrNetlistViewCreateNetScalar

DECLARATION

```
cfidrNetScalarIdT cfidrNetlistViewCreateNetScalar(
cfidrNetlistViewIdT owner,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function creates a NetScalar object owned by the NetlistView specified via *owner*. The NetScalar is referred to by *name*.

RATIONALE

This function only allows NetScalars to be created and owned by the NetlistView. The NetScalar can be subsequently contained in a NetBundle.

RETURN VALUE

The return value is a cfidrNetScalarIdT referring to the NetScalar just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the NetlistView owning the NetScalar to be created.

name (input) The string representing the name of the NetScalar.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a NetlistView.

CFIDR_READ_ONLY:
the owner View has not been opened for update.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_NAME_IN_USE:
the name parameter specifies a name already in use in the same name scope but not for a NetScalar.

CFIDR_OBJECT_ALREADY_EXISTS:
a NetScalar with the same name already exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

A Net (NetScalar, NetBundle, or NetBus) with the given *name* cannot already be owned by the NetlistView specified by *owner*.

POST-CONDITIONS

The NetScalar created by this function can be later inserted into one or more NetBundle(s).

The *IsGlobal* attribute is initially set to CFIDR_FALSE. Use the function **cfidrNetScalarSetIsGlobal()** to change it.

REFERENCE

cfidrNetScalarSetIsGlobal()
cfidrNetlistViewGetNets()
cfidrNetBundleGetNets()
cfidrNetBundleInsertNet()
cfidrNetBundleRemoveNet()

5.13.6 cfidrNetlistViewCreatePortBundle

DECLARATION

cfidrPortBundleIdT cfidrNetlistViewCreatePortBundle(
cfidrNetlistViewIdT owner,
cfidrStringT name,
cfidrInt32T position,
cfidrErrorT *error)

DESCRIPTION

This function creates a PortBundle object owned by the NetlistView specified via *owner*.

RETURN VALUE

The return value is a cfidrPortBundleIdT referring to the PortBundle object just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the NetlistView owning the PortBundle being created.

name (input) The string representing the name of the PortBundle being created.

position (input) The integer specifying the position of the PortBundle in the list of Ports owned by *owner*. A *position* value less than 0 or greater than or equal to the number of ports indicates the new PortBundle is put at end of the list. Any other value, including 0, indicates the actual position in the list.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:

owner is not the OID of a NetlistView.

CFIDR_READ_ONLY:
the owner View has not been opened for update.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_NAME_IN_USE:
the name parameter specifies a name already in use in the same name scope but not for a PortBundle.

CFIDR_OBJECT_ALREADY_EXISTS:
a PortBundle with the same name already exists for this *owner*.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

A Port named *name* that is owned by the same NetlistView as specified by *owner* cannot already exist.

POST-CONDITIONS

The PortBundle is created with no members initially. Later, Ports or PortBundles may either be created in or have their owner changed to be this PortBundle. Furthermore, this PortBundle may later have its owner changed to a PortBundle instead of the NetlistView.

The PortBundle is created and added to the list of Ports owned by the NetlistView specified by *owner* at the position specified by position. If position = 0, the new PortBundle will be added as the first member of the list. If position is less than 0 or greater than or equal to the number of members in the list, the new PortBundle is appended to the end of the list. Otherwise, the PortBundle is inserted at the specified position (0 based). The position of all Ports whose positions were greater than or equal to position, will have their position incremented by 1. For example, presume a NetlistView initially has Ports:
A, B, C0, 1, 2.

Adding a new member X at position 1 results in:
A, X, B, C0, 1, 2
Note that B had position 1 before and has position 2 after X is created.

Adding X at position 0 causes X to be at the beginning of the list:
X, A, B, C0, 1, 2

Adding X at an invalid position such as -1 (or any value greater than 4) causes creation of the new member at the end of the list:
A, B, C0, 1, 2, X

REFERENCE

cfidrPortBundleCreatePortBundle()
cridrPortBundleCreatePortScalar()
cfidrPortBundleGetOwner()
cfidrPortBundleGetPorts()
cfidrPortSetOwnerPortBundle()
cfidrPortSetOwnerView()
cfidrNetlistViewGetPorts()

5.13.7 cfidrNetlistViewCreatePortBus

DECLARATION

cfidrPortBusIdT cfidrNetlistViewCreatePortBus(
cfidrNetlistViewIdT owner,
cfidrStringT name,
cfidrInt32T position
cfidrInt32T start
cfidrInt32T step
cfidrErrorT *error)

DESCRIPTION

This function creates a PortBus object located at the position *position* in the list of Ports owned by the NetlistView specified via *owner*. The PortBus *Name* attribute is set to *name*. The PortBus *Start* attribute is set to *start*. The PortBus *Step* attribute is set to *step*.

RETURN VALUE

The return value is a cfidrPortBusIdT referring to the PortBus object just created. If an error occurs, a Null OID is returned.

PARAMETERS

owner (input) The OID of the NetlistView owning the PortBus being created.

name (input) The string representing the name of the PortBus being created.

position (input) The integer specifying the position of the PortBus in the list of Ports owned by *owner*. A *position* value less than 0 or greater than or equal to the number of ports indicates the new PortBus is put at end of the list. Any other value, including 0, indicates the actual position in the list.

start (input) The Int32 defining the index of position 0 in the bus.

step (input) The Int32 defining the difference in indexes of positions n and n+1 in the bus. This number must be non-zero.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating

the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a NetlistView.

CFIDR_READ_ONLY:
the owner View has not been opened for update.

CFIDR_INVALID_NAME:
the *name* parameter is not a valid string or is not a legal name.

CFIDR_NAME_IN_USE:
the name parameter specifies a name already in use in the same name scope but not for a PortBus.

CFIDR_OBJECT_ALREADY_EXISTS:
a PortBus with the same name already exists.

CFIDR_INVALID_VALUE
Step is 0.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

A Port (PortScalar, PortBundle, or PortBus) with the given *name* cannot already exist in the NetlistView specified by *owner*.

POST-CONDITIONS

The newly created PortBus will initially have no members and a size of 0.

The PortBus created by this function can later be changed to have a PortBundle as its owner using **cfidrPortBundleSetOwnerPortBundle()**.

The *Start* and *Step* attributes may be changed later by use of the **cfidrPortBusSetStart()** and **cfidrPortBusSetStep()** functions.

REFERENCE

cfidrPortBundleGetSize()
cfidrPortBundleGetNets()
cfidrPortBundleSetOwnerPortBundle()
cfidrPortBundleSetOwnerView()
cfidrPortBusGetStart()
cfidrPortBusGetStep()
cfidrPortBusSetStart()
cfidrPortBusSetStep()
cfidrNetlistViewGetPortBundles()

5.13.8 cfidrNetlistViewCreatePortScalar

DECLARATION

cfidrPortScalarIdT cfidrNetlistViewCreatePortScalar(
cfidrNetlistViewIdT owner,
cfidrStringT name,
cfidrInt32T position,
cfidrPortDirectionT direction,
cfidrErrorT *error)

DESCRIPTION

This function creates a PortScalar object owned by the NetlistView specified via *owner* and located at *position* in the list of Ports owned by *owner*. The PortScalar *Name* attribute is set to *name*. The PortScalar *Direction* attribute is set to *direction*.

RATIONALE

The PortScalar is created with all required attributes set including the derived attribute of position.

RETURN VALUE

The return value is a cfidrPortScalarIdT referring to the PortScalar just created. If an error occurs a Null OID is returned.

PARAMETERS

owner (input) The OID of the NetlistView owning the PortScalar to be created.

name (input) The string representing the name of the PortScalar.

position (input) The integer specifying the position of the PortScalar in the list of Ports owned by *owner*. A *position* value less than 0 or greater than or equal to the number of ports indicates the new Port is put at end of the list. Any other value, including 0, indicates the actual position in the list.

direction (input) The *Direction* attribute to assign to the created PortScalar.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating

the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a NetlistView.

CFIDR_READ_ONLY:
the owner View has not been opened for update.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_NAME_IN_USE:
the name parameter specifies a name already in use in the same name scope but not for a PortScalar.

CFIDR_OBJECT_ALREADY_EXISTS:
a PortScalar with the same *name* already exists for *owner*.

CFIDR_INVALID_DIRECTION:
direction is not a legal PortDirection.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* argument must conform to the character set rules for name strings.

A Port (PortScalar, PortBundle, or PortBus) with the given *name* cannot already be owned by the NetlistView specified by *owner*. However, a Port with the same name may exist in a PortBundle. This is allowed since the scope of Port names is the owning NetlistView or the owning PortBundle.

The *owner* of the PortScalar created by this function can be later be changed to a PortBundle provided no other member of that PortBundle has the same *name*.

POST-CONDITIONS

The PortScalar is created and added to the list of Ports owned by the NetlistView specified by *owner* at the position specified by position. If position = 0, the new PortScalar will be added as the first member of the list. If position is less than 0 or greater than or equal to the number of members in the list, the new PortScalar is appended to the end of the list. Otherwise, the PortScalar is inserted at the specified position (0 based). The position of all Ports whose positions were greater than or equal to position, will have their position incremented by 1. For example, presume a NetlistView has Ports:

A, B, C0, 1, 2.

Adding a new member X at position 1 results in:
A, X, B, C0, 1, 2
Note that B had position 1 before and has position 2 after X is created.

Adding X at position 0 causes X to be at the beginning of the list:
X, A, B, C0, 1, 2

Adding X at an invalid position such as -1 (or any value greater than 4) causes creation of the new member at the end of the list:
A, B, C0, 1, 2, X

REFERENCE

cfidrPortBundleCreatePortBundle()
cridrPortBundleCreatePortScalar()
cfidrPortBundleGetOwner()
cfidrPortBundleGetPorts()
cfidrPortSetOwnerPortBundle()
cfidrPortSetOwnerView()
cfidrNetlistViewCreatePortBundle()
cfidrNetlistViewGetPorts()

5.13.9 cfidrNetlistViewFindInstByName

DECLARATION

```
cfidrInstIdT cfidrNetlistViewFindInstByName(
cfidrNetlistViewIdT netlistView,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the Inst with the specified name that is within the Insts relationship defined for the NetlistView specified by netlistView.

RETURN VALUE

The return value is a cfidrInstIdT referring to the Inst just found. If an error occurs, a Null OID is returned.

PARAMETERS

netlistView (input) The OID of the NetlistView from which the Inst is to be found.

name (input) The name of the Inst.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netlistView is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netlistView is not the OID of a NetlistView.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no Inst with the specified name exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

5.13.10 cfidrNetlistViewFindNetByName

DECLARATION

```
cfidrNetIdT cfidrNetlistViewFindNetByName(
cfidrNetlistViewIdT netlistView,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the Net (either NetBundle or NetScalar) with the specified name that is within the Nets relationship defined for the NetlistView specified by netlistView. All Nets are scoped to the NetlistView, therefore Nets contained in NetBundles are also searched by this function.

RETURN VALUE

The return value is a cfidrNetIdT referring to the Net just found. If an error occurs, a Null OID is returned.

PARAMETERS

netlistView (input) The OID of the NetlistView from which the Net is to be found.

name (input) The name of the Net.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netlistView is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netlistView is not the OID of a NetlistView.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no Net with the specified name exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

5.13.11 cfidrNetlistViewFindPortByName

DECLARATION

```
cfidrPortIdT cfidrNetlistViewFindPortByName(
cfidrNetlistViewIdT netlistView,
cfidrStringT name,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the Port (either PortBundle or PortScalar) with the specified name that is within the Ports relationship defined for the NetlistView specified by netlistView. All Ports are scoped to their Owner (either NetlistView or PortBundle), therefore Ports contained in PortBundles are not searched by this function.

RETURN VALUE

The return value is a cfidrPortIdT referring to the Port just found. If an error occurs, a Null OID is returned.

PARAMETERS

netlistView (input) The OID of the NetlistView from which the Port is to be found.

name (input) The name of the Port.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netlistView is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netlistView is not the OID of a NetlistView.

CFIDR_INVALID_NAME:
the name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no Port with the specified name exists.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

5.13.12 cfidrNetlistViewGetInsts

DECLARATION

```
cfidrInstsIdT cfidrNetlistViewGetInsts(
cfidrNetlistViewIdT netlistView,
cfidrErrorT *error)
```

DESCRIPTION

This function initiates a traversal of all the Insts at the end of the *Insts* relationship for the NetlistView specified by *netlistView*.

RETURN VALUE

The return value is a cfidrInstsIdT referring to an Iterator ID that iterates over Insts. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the **cfidrIterNextInst()** function returns a Null OID.

PARAMETERS

netlistView (input) The OID of a NetlistView for which the *Insts* relationship is to be traversed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
netlistView is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
netlistView is not the OID of a NetlistView.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The **cfidrIterNextInst()** function returns the next Inst OID in the *Insts* relationship.

If no objects are currently present in the *Insts* relationship, an error is not returned but the first call to **cfidrIterNextInst()** returns a Null OID.

REFERENCE

cfidrIterNextInst()

5.13.13 cfidrNetlistViewGetNets

DECLARATION

```
cfidrNetsIdT cfidrNetlistViewGetNets(
cfidrNetlistViewIdT view,
cfidrIterModeT mode,
cfidrErrorT *error)
```

DESCRIPTION

This function initiates a traversal of all of the Nets (NetBundles, NetBusses, or NetScalars) at the end of the *Nets* relationship defined for the View specified by *view*.

RETURN VALUE

The return value is a cfidrNetsIdT referring to an Iterator ID that iterates over Nets. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextNet() function returns a Null OID.

PARAMETERS

view (input) The OID of a NetlistView for which the *Nets* relationship is to be traversed.

mode (input) The mode that determines which Nets are returned. The current valid values are CFIDR_ITER_SCALARS, CFIDR_ITER_BUNDLES, CFIDR_ITER_TOP, and CFIDR_ITER_ALL. See the Pre-Conditions below for more detail on how these modes behave.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
view is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
view is not the OID of a NetlistView.

CFIDR_INVALID_ITERMODE:
mode is not a valid IterMode.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Returns all Nets owned by the NetlistView. The Nets returned depends on the *mode* argument specified.

mode=CFIDR_ITER_TOP
Returns all those Nets (NetBundles, NetBusses, or NetScalars) which are not contained in a NetBundle.

mode=CFIDR_ITER_SCALARS
Returns all the NetScalars owned by the NetlistView, regardless of whether they are contained in a NetBundle or not. No NetBundles are output with this mode, only NetScalars. This mode lets the application effectively see the View as a scalar netlist.

mode=CFIDR_ITER_BUNDLES
Returns all the NetBundles owned by the NetlistView. No NetScalars are output.

mode=CFIDR_ITER_ALL
Returns all the Nets (NetBundles, NetBusses, or NetScalars) owned by the NetlistView. This mode differs from CFIDR_ITER_TOP in that NetScalars contained in a Bundle are also returned. This mode is a combination of the SCALARS and BUNDLES modes.

POST-CONDITIONS

The **cfidrIterNextNet()** function returns the next Net OID in the *Nets* relationship. The next Net returned must obey the return mode as specified via the *mode* argument.

If no objects are currently present in the *Nets* relationship, an error is not returned but the first call to

cfidrIterNextNet() returns a Null OID.

REFERENCE

cfidrIterNextNet()

5.13.14 cfidrNetlistViewGetPorts

DECLARATION

```
cfidrPortsIdT cfidrNetlistViewGetPorts(
cfidrNetlistViewIdT view,
cfidrIterModeT mode,
cfidrErrorT *error)
```

DESCRIPTION

This function initiates a traversal of all of the Ports (PortBundles, PortBusses, or PortScalars) at the end of the *Ports* relationship defined for the NetlistView specified by *view*.

RETURN VALUE

The return value is a cfidrPortsIdT referring to an Iterator ID that iterates over Ports. A valid Iterator ID is always returned, even when an error occurs. In the case of an error, calling the cfidrIterNextPort() function returns a Null OID.

PARAMETERS

view (input) The OID of a NetlistView for which the *Ports* relationship is to be traversed.

mode (input) The mode that determines which Ports are returned. The current valid values are CFIDR_ITER_SCALARS, CFIDR_ITER_BUNDLES, CFIDR_ITER_TOP, and CFIDR_ITER_ALL. See the Pre-Conditions below for more detail on how these modes behave.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
view is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
view is not the OID of a NetlistView.

CFIDR_INVALID_ITERMODE:
mode is not a valid IterMode.

CFIDR_INTERNAL_SYSTEM_ERROR:

some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

Returns all or some of the Ports directly or indirectly owned by the NetlistView.

Ports in a View each have a unique position in the list of Ports owned by the NetlistView. The positions in this list are numbered from 0 for the "leftmost" or first position monotonically increasing by 1 up to one less than the size of the list. The order in which the Ports are returned is by increasing position starting with 0. Note that the Position of a given Port at a given time is determined by the various operations that have been performed on this list including:
(1) the position specified when a Port is created in a View,
(2) the position specified when a Port's owner is changed to the View, and
(3) the appearance (from inside PortBundles) and disappearance (into PortBundles) of Ports at or to the "left" of the position of the given Port.

The Ports returned depends on the *mode* argument specified.

mode=**CFIDR_ITER_TOP**
Returns all those Ports (PortBundles, PortBusses, or PortScalars) which are owned directly by *view* (i.e., not owned by a PortBundle).

mode=**CFIDR_ITER_SCALARS**
Returns all the PortScalars in *view*, regardless of whether they are contained in a PortBundle or not. No PortBundles are output with this mode.

mode=**CFIDR_ITER_BUNDLES**
Returns all the PortBundles owned directly or indirectly by *view*. No PortScalars are output with this mode.

mode=**CFIDR_ITER_ALL**
Returns all the Ports (PortBundles, PortBusses, or PortScalars) directly or recursively in *view*. This mode differs from CFIDR_ITER_TOP in that Ports contained indirectly in PortBundles are also returned.

The order for all modes except CFIDR_ITER_TOP is determined by depth-first recursion.

The Port name scoping rules state that Port names need only be unique within their owner (either the View or a PortBundle). Thus, in any mode except CFIDR_ITER_TOP two Ports with the same name may be returned that are completely different objects. The function **cfidrObjectIsSame()** can be used to determine if any two ports are the same.

POST-CONDITIONS

The **cfidrIterNextPort()** function returns the next Port OID in the *Ports* relationship.

If no objects are currently present in the *Ports* relationship, an error is not returned but the first call to

cfidrIterNextPort() returns a Null OID.

REFERENCE

cfidrIterNextPort()
cfidrPortSetOwnerPortBundle()
cfidrPortSetOwnerView()

5.13.15 cfidrViewGetOwner

DECLARATION

```
cfidrCellIdT cfidrViewGetOwner(
cfidrViewIdT view,
cfidrErrorT *error)
```

DESCRIPTION

This function returns the Cell at the end of the *Owner* relationship for the View specified by *view*. The *Owner* relationship is defined when an object is created and cannot be modified.

RETURN VALUE

The return value is a cfidrCellIdT referring the Cell at the end of the *Owner* relationship. If an error occurs, a Null OID is returned.

PARAMETERS

view (input) The OID of the View whose *Owner* relationship is to be followed.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
view is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
view is not the OID of a View.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrCellGetViews()

5.13.16 cfidrViewGetViewType

DECLARATION

cfidrStringT cfidrViewGetViewType(
cfidrViewIdT view,
cfidrErrorT *error)

DESCRIPTION

This function returns the *ViewType* attribute of the View specified by *view*.

RETURN VALUE

The return value is a cfidrStringT representing the value of the *ViewType* attribute. On error the string "" is returned.

PARAMETERS

view (input) The OID of the View whose *ViewType* attribute is desired.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
view is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
view is not the OID of a View.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrCellCreateEncapsulatedView()
cfidrCellCreateNetlistView()

5.13.17 cfidrViewPurge

DECLARATION

```
cfidrVoidT cfidrViewPurge(
cfidrViewIdT view,
cfidrErrorT *error)
```

DESCRIPTION

This function purges the View specified by *view*. The purge operation removes any changes made to an object since the last save or open. See the POST-CONDITIONS for more specific information.

PARAMETERS

view (input) The OID of the View to purge. The View hierarchy owned by *view* is also purged.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
view is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
view is not the OID of a View.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

A View can be purged regardless of the access mode of any object.

POST-CONDITIONS

Any OIDs referring to the purged *view* are made unusable. They may be reused to refer to other Views at some point, so the application cannot assume that if the purged View is again made available (via open, create, etc.), that it will have the same OID.

It is neither required nor precluded that whatever encapsulated information referenced by the EncapsulatedView be purged since this can be system dependent.

The effect of purging a NetlistView is to also purge the entire object hierarchy owned by that NetlistView.

Policy: The purge operation removes any changes made to the View since the last **save** or **open**. The View then becomes unavailable and its OID becomes unusable. Opening the View again assigns the

View an OID and makes it available for updating.

An example implementation that provides this behavior would make an in-memory copy of the view when it was opened. Any changes to the view, or to any of its contained objects, would be made to the in-memory copy of the view. The effect of a "purge" is to delete the in-memory copy of the view, leaving the on-disk persistent view data unchanged.

Policy: Similar to **open**, **purge** also recursively walks the ownership hierarchy of Objects rooted by the specified View and purges all the Objects in this hierarchy. Unlike **open**, **purge** continues walking the ownership hierarchy when it encounters an Object which can be purged individually. Thus, **purge** behaves similarly to **save** in this respect.

Rationale: Consider the information model and take into account the policy that only Libraries and Views can be purged. Purging a View removes the View's Objects (Nets, Ports, Insts, and Props). Purging a Lib not only purges the Lib and Cells, it also purges all the Views as well.

Rationale: When opening a Lib, opening all the Views automatically is probably undesirable in most situations. However, when purging a Lib, it does not make sense to only purge the Lib and Cells and not the Views.

Policy: Purging a NetlistView "A" (and its owned Ports) that is referenced in another NetlistView "B" via the *Describer* relationship of an Inst (and of its PortInsts) only breaks the *Describer* relationships from the Inst (to "A") and the PortInsts. The Inst and PortInsts are not themselves purged nor is NetlistView "B".

REFERENCE

cfidrLibSave()
cfidrLibPurge()
cfidrViewSave()

5.13.18 cfidrViewSave

DECLARATION

```
cfidrVoidT cfidrViewSave(
cfidrViewIdT view,
cfidrErrorT *error)
```

DESCRIPTION

This function saves the View specified via *view*. The effect of saving a View is to also save the entire Object hierarchy owned by the saved View. Saving a View makes all updates to objects in the View's hierarchy persistent. If the View has been opened for CFIDR_READ, changes to the Views contained in the View that are open for CFIDR_UPDATE will be made persistent.

PARAMETERS

view (input) The OID of the View to save. The Objects owned by *view* are also saved.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
view is not a usable OID.

CFIDR_INVALID_OBECTTYPE:
view is not the OID of a View.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

Views cannot be saved completely independently of Libs because of the issue of referential integrity for the View's Cell's catalog entry in the Lib. Thus when a View is saved, some data associated with the owning Lib of the View will also likely have to be saved.

The Lib must also be saved such that the Cell with this View is saved. Depending on the implementation this may already be true and thus no further Lib save is needed. However, if this is a newly-created View since the last save of the Lib with the Cell containing this view, a partial save will be necessary to ensure that the Cell reference to this view is now persistent.

Other newly created but not yet saved Views are not included in the saved Lib until they are explicitly saved. However those Views still appear in "memory" unless the Cell containing them is purged.

Once a View is saved, it is known and available for opening across session boundaries until deleted via **cfidrObjectDestroy()**.

Saving does not invalidate OIDs referring to the saved View.

REFERENCE

cfidrObjectDestroy()
cfidrPIQuit()

 Table of Contents  Next Chapter

Table of Contents Previous Chapter

6 View Selection

6.1 View Selection Overview

View Selection allows selection between different, alternative but substitutable views in a design hierarchy. While the Describer attribute of an Inst is always a specific Cell and View (namely, the *describerView* parameter supplied to **cfidrViewCreateInst()** when the Inst is created), it is possible when traversing a design hierarchy to choose a different, equivalent View. Certainly as long as the Ports of the alternative describer View have the same name, number, and directions of the PortInsts on the Inst, it is reasonable to choose the alternative View.

> NOTE: While it is certainly *allowed*, there is no *requirement* specified in DR 1.0 that the Ports of multiple NetlistViews of Cell must match in name, number, direction, or types. Furthermore, it is by definition outside of the scope of DR 1.0 to specify what sort of "ports" are in an "outside" View that is encapsulated by a cfidrEncapsulatedView or to specify how those "ports" correspond to the Ports in any particular cfidrNetlistView.

View Selection provides rules for choosing alternatives that make it possible for the user to insure that the design hierarchy is traversed in the same manner by two different tools. View Selection does not, at this time guarantee that the Views selected are equivalent to the original describer. That is up to the user of the system and, in particular, the creator of the selector objects. As with the other features of the CFI DR-PI, a mechanism is defined but methodology and policies are left to the CAD system integrator.

It is anticipated that the typical use of this mechanism will be to selectively specify the use of substructure versus available behavioral descriptions when a design is being expanded for simulation. As such, it will work naturally with the Encapsulated View mechanism, allowing either choices of multiple connectivity models (in the scope of the CFI 1.0 DR Information Model and Programming Interface) or alternative "foreign" models which must be manually synchronized to the (interface) Ports in the other connectivity views.

This overview first describes the additional objects needed for view selection and then describes the types of routines needed for view selection. It is followed by a more formal description of the Information Model and then by the function descriptions for the DR-PI.

6.1.1 Disclaimer

This is a very preliminary version of a more general capability for ViewSelection anticipated to be in CFI 2.0 Design Representation. That version of CFI DR is expected to support versioning and rebinding of View references and therefore a more general concept of instantiation. There will also likely be a CFI-defined standard mechanism for mapping between Views and the objects inside of them.

6.1.2 View Selection Objects

The object types that are used to support view selection are:

1. **SelectorSet**---this object contains a set of prioritized ViewSelectors and a set of prioritized LibrarySelectors. The ViewSelectors are used to specify which Views should be selected and the LibrarySelectors are used to specify where to look for Views.
   LibrarySelector---this specifies which Libraries (Libs) may contain possible Cells to be selected.
   ViewSelector---this specifies which View to select. A ViewSelector may be either a ViewNameSelector or a ViewTypeSelector. A ViewSelector consists of source attributes and destination attributes. The source attributes specify that a ViewSelector should only be applied in selecting a View for a Cell if the Cell matches the source attributes. Currently the source attributes are LibName and CellName. The destination attributes specify which ViewName or ViewType to select.
   ViewNameSelector---this is a subtype of selector which contains selectors that specify particular ViewNames upon which to select. It may also specify a source LibName and/or source CellName which must match the *Name* attribute of the Cell for which we are selecting a View in order for the rule to be applied.
   ViewTypeSelector---this is a subtype of a selector which contains selectors that specify particular ViewTypes to select upon. It may also specify a source LibName and/or source CellName which must match the *Name* attribute of the Cell for which we are selecting a View in order for the rule to be applied.

6.1.3 View Selection Operations

View Selection is accomplished with a set of LibrarySelectors which specify where Cells and their Views may be found and a set of ViewSelectors which specify which Cells and their Views should be selected. The selection routine **cfidrSelectorSetSelectView()** looks for the View in the Libraries (Libs) specified by the LibrarySelectors. **cfidrSelectorSetSelectView()** starts at the first ViewSelector and sees whether it selects a particular View. It continues through the ViewSelectors in order of creation until it finds a Selector which selects exactly one View. When **cfidrSelectorSetSelectView()** finds a ViewSelector that specifies a particular View it returns that cfidrViewIdT to the calling program.

There are four types of view selection routines, CREATION, PERSISTENCY, ATTRIBUTE ACCESS, AND SELECTION:

1. CREATION: The creation routines create the SelectorSets and the various kinds of selectors in the SelectorSets.
   cfidrPICreateSelectorSet();
   cfidrSelectorSetCreateLibrarySelector();
   cfidrSelectorSetCreateViewNameSelector();
   cfidrSelectorSetCreateViewTypeSelector();
   PERSISTENCY:
   cfidrPIOpenSelectorSet();
   cfidrSelectorSetPurge();
   cfidrSelectorSetSave();
   ATTRIBUTE ACCESS: The attribute access routines access the created SelectorSets and selectors. It is currently not possible to modify the attributes of an existing selector.
   cfidrIterNextLibrarySelector();
   cfidrIterNextViewSelector();
   cfidrLibrarySelectorGetLibName();
   cfidrViewNameSelectorGetCellName();

cfidrViewNameSelectorGetLibName();
cfidrViewNameSelectorGetViewName();
cfidrViewTypeSelectorGetCellName();
cfidrViewTypeSelectorGetLibName();
cfidrViewTypeSelectorGetViewType();
SELECTION: This routine is used to select a particular view given a SelectorSet and a cell.
cfidrSelectorSetSelectView();

6.1.4 View Selection Examples

1. ViewNameSelector with specified LibName and CellName

```
SrcLibName:mylib

SrcCellName:AND2

DestViewName:beh
```

This ViewNameSelector selects any View named "beh" for a Cell with the CellName "AND2" and a Lib with LibName "mylib" if a View with the name "beh" exists for the Cell "AND2" in any of the Libraries specified by the LibrarySelectors.

ViewNameSelector with no specified LibName and CellName

```
SrcLibName:

SrcCellName:

DestViewName:netlist
```

If the user has Libraries (Libs) whose *Name* attributes are: "LibA", "LibB", and "LibC" (created in this order) specified in the LibrarySelector and "LibA" has a View whose *Name* attribute is "schematic," "LibB" has a View named "netlist," and "LibC" has Views named "netlist" and "schematic" for the Cell this rule would return the "LibB" View named "netlist."

ViewNameSelector with specified CellName but no specified LibName.

```
SrcLibName:

SrcCellName:Adder

DestViewName:schematic
```

This works like example 2 except that it is only applied to Cells whose *Name* attribute is "Adder," this time looking for Views whose *Name* attribute is "schematic."

ViewNameSelector with specified LibName but no specified CellName.

```
SrcLibName:VendorLib

SrcCellName:
```

```
DestViewName:VendorSim
```

This works like example2 except that it is only used with Cells whose Lib *Name* attribute is "VendorLib."

6.2 View Selection Information Model

6.2.1 EXPRESS-G

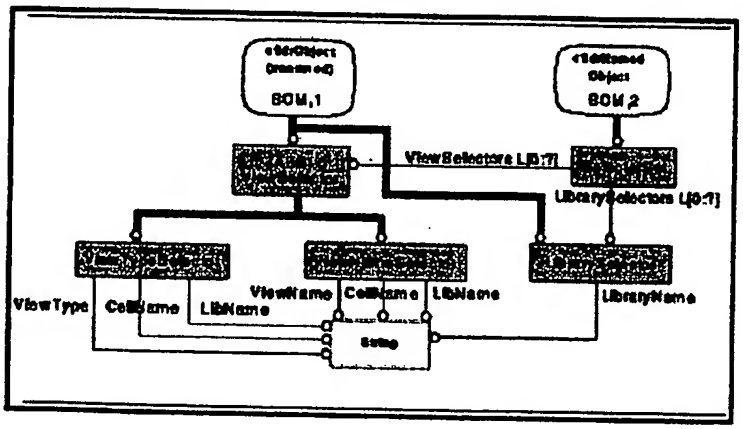

Figure 7.1 View Selection Information Model

6.2.2 Entity: SelectorSet

```
ENTITY cfidrSelectorSet
  SUBTYPE OF (cfidrNamedObject);
  ViewSelectors: LIST[0:?] of
      UNIQUE cfidrViewSelector;
  LibrarySelectors: LIST[0:?] of
      UNIQUE cfidrLibrarySelector;
END_ENTITY;
```

6.2.3 Entity: ViewSelector

```
ENTITY cfidrViewSelector
  SUBTYPE OF (cfidrObject)
  ABSTRACT SUPERTYPE OF (ONEOF
  (cfidrViewNameSelector,  cfidrViewTypeSelector));
  Owner: cfidrSelectorSet
```

```
END_ENTITY;
```

6.2.4 Entity: ViewNameSelector

```
ENTITY cfidrViewNameSelector
  SUBTYPE OF (cfidrViewSelector);
  LibName: OPTIONAL cfidrStringT;
  CellName: OPTIONAL cfidrStringT;
  ViewName: cfidrStringT;
END_ENTITY;
```

The LibName and CellName are optional attributes which are used to limit the application of a particular Selector to only those cells whose LibName and/or ViewName matches the cell whose view selection is desired.

6.2.5 Entity: ViewTypeSelector

```
ENTITY cfidrViewTypeSelector
  SUBTYPE OF (cfidrSelector);
  LibName:  OPTIONAL cfidrStringT;
  CellName: OPTIONAL cfidrStringT;
  ViewType: cfidrStringT;
END_ENTITY;
```

The LibName and CellName are optional attributes which are used to limit the application of a particular Selector to only those cells whose LibName and/or ViewName matches the cell whose view selection is desired.

6.2.6 Entity: LibrarySelector

```
ENTITY cfidrLibrarySelector
  SUBTYPE OF (cfidrObject);
  libraryName: cfidrStringT;
  Owner:     cfidrSelectorSet
END_ENTITY;
```

6.3 View Selection Data Types

Object Identifiers

```
typedef cfidrObjectIdT    cfidrSelectorSetIdT;
typedef cfidrObjectIdT    cfidrViewSelectorIdT;
typedef cfidrObjectIdT    cfidrViewNameSelectorIdT;
typedef cfidrObjectIdT    cfidrViewTypeSelectorIdT;
typedef cfidrObjectIdT    cfidrLibrarySelectorIdT;
```

Iterator Identifiers

```
typedef cfidrIterIdT    cfidrViewSelectorsIdT;
typedef cfidrIterIdT    cfidrLibrarySelectorsIdT;
```

Types of Objects

Additions to cfidrObjectTypeT

```
typedef enum {
 = 0,    CFIDR_UNDEFINED_OBJECTTYPE
 = 1,    CFIDR_LIB
 = 2,    CFIDR_CELL
 = 3,    CFIDR_PORTBUNDLE
 = 4,    CFIDR_PORTBUS
 = 5,    CFIDR_PORTSCALAR
 = 6,    CFIDR_INST
 = 7,    CFIDR_PORTINSTBUNDLE
 = 8,    CFIDR_PORTINSTBUS
 = 9,    CFIDR_PORTINSTSCALAR
 = 10,    CFIDR_NETBUNDLE
 = 11,    CFIDR_NETBUS
 = 12,    CFIDR_NETSCALAR
 = 13,    CFIDR_PROP
 = 14,    CFIDR_NETLISTVIEW
 = 15,    CFIDR_ENCAPSULATEDVIEW
 = 16,    CFIDR_SELECTORSET
 = 17,    CFIDR_LIBSELECTOR
 = 18,    CFIDR_VIEWNAMESELECTOR
 = 19,    CFIDR_VIEWTYPESELECTOR
 = 20     CFIDR_MAX_OBJECTTYPE
} cfidrObjectTypeT;
```

6.4 View Selection Programming Interface

6.4.1 cfidrIterNextLibrarySelector

DECLARATION

cfidrLibrarySelectorIdT cfidrIterNextLibrarySelector(
cfidrLibrarySelectorsIdT iterator,
cfidrErrorT *error)

DESCRIPTION

The function returns another LibrarySelector object from the Iterator ID specified via *iterator*.

RETURN VALUE

If there are any more LibrarySelectors in the *iterator*, the return value is an OID of type **cfidrLibrarySelectorIdT** for the next LibrarySelector. If an error occurs or there are no more LibrarySelectors, a Null OID is returned. If there are no more LibrarySelectors this is not an error.

PARAMETERS

iterator (input) The Iterator ID of an Iterator over LibrarySelector objects.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iterator is not a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iterator is not an Iterator of LibrarySelectors.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

6.4.2 cfidrIterNextSelectorSet

DECLARATION

```
cfidrSelectorSetIdT cfidrIterNextSelectorSet(
cfidrSelectorSetsIdT iter,
cfidrErrorT *error)
```

DESCRIPTION

This function returns another Selector Set object from the Iterator ID specified via iter.

RETURN VALUE

The return value is a cfidrSelectorSet referencing the Selector Set object just iterated.

If an error occurs or there are no more Selector Sets to iterate, a Null OID is returned. If there are no more Selector Sets to iterate, this is not an error.

PARAMETERS

iter (input) The Iterator ID representing an Iterator of Selector Set objects.

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iter is not the ID of a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iter is not the ID of an Iterator of Selector Set objects.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The iter argument must have been returned via a previous call to cfidrPIGetSelectorSets().

POST-CONDITIONS

If the Selector Set is not already open, this function will attempt to open it. If the open fails, the error return for this function will be the error return of the open. The iteration sequence can continue after a failed Selector Set open.

REFERENCE

cfidrPIGetSelectorSets()

6.4.3 cfidrIterNextViewSelector

DECLARATION

cfidrViewSelectorIdT cfidrIterNextViewSelector(
cfidrViewSelectorsIdT iterator,
cfidrErrorT *error)

DESCRIPTION

The function returns another ViewSelector object from the Iterator ID specified via *iterator*.

RETURN VALUE

If there are any more ViewSelectors in the *iterator*, the return value is an OID of type **cfidrViewSelectorIdT** for the next ViewSelector. If an error occurs or there are no more ViewSelectors, a Null OID is returned. If there are no more ViewSelectors this is not an error.

PARAMETERS

iterator (input) The Iterator ID representing an Iterator ViewSelector objects.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INVALID_ITER:
iterator is not a valid Iterator.

CFIDR_INVALID_ITER_TYPE:
iterator is an Iterator of ViewSelectors.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

6.4.4 cfidrLibrarySelectorGetLibraryName

DECLARATION

cfidrStringT cfidrLibrarySelectorGetLibraryName(
cfidrLibrarySelectorIdT librarySelector,
cfidrErrorT *error)

DESCRIPTION

Returns the *LibraryName* attribute of the LibrarySelector specified in *librarySelector*.

RETURN VALUE

The return value is of type **cfidrStringT** and is the value of the *LibraryName* attribute of *librarySelector*. On error a Null string (" ") is returned.

PARAMETERS

librarySelector (input) The OID of the LibrarySelector whose *LibraryName* attribute is desired.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
librarySelector is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
librarySelector is not the OID of a LibrarySelector.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

6.4.5 cfidrPICreateSelectorSet

DECLARATION

cfidrSelectorSetIdT cfidrPICreateSelectorSet(
cfidrStringT name,
cfidrErrorT *error)

DESCRIPTION

Creates a SelectorSet with the specified *name*. The SelectorSet may contain a prioritized/ordered list of ViewSelectors and LibrarySelectors. The Selectors are prioritized based on the order of creation. The first Selector created will have the highest priority. The second Selector created will have the next highest priority.

RETURN VALUE

The return value is an OID of type **cfidrSelectorSetIdT** for the SelectorSet just created. If an error occurs a Null OID is returned.

PARAMETERS

name (input) The string representing the name of the SelectorSet to be created.

error (output) The error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_INVALID_NAME
the *name* parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_ALREADY_EXISTS:
a SelectorSet with the same *name* already exists.

CFIDR_INTERNAL_SYSTEM_ERROR
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

The *name* of the SelectorSet must conform to the character set for name strings. No SelectorSet with the same name may exist.

The system implementing the DR-PI is responsible for knowing about all SelectorSets created and saved in previous sessions or created in the current session when determining whether the SelectorSet already exists. The SelectorSet need not be opened to be considered as existing.

Each system implementing the DR-PI is responsible for creating and maintaining the SelectorSet's data-storage area. The application using the PI need not have any prior knowledge of where the data physically resides.

If cfidrPIINIT() has not been called previously, it is called automatically before this function.

POST-CONDITIONS

A SelectorSet is created with no ViewSelectors or LibrarySelectors.

A created SelectorSet is not persistent across session boundaries until it is saved.

6.4.6 cfidrPIGetSelectorSets

DECLARATION

```
cfidrSelectorSetsIdT cfidrPIGetSelectorSets(
cfidrErrorT *error)
```

DESCRIPTION

This function initiates a traversal of all the Selector Sets known by the DR-PI.

RETURN VALUE

The return value is a cfidrSelectorSetsIdT referring to an Iterator ID that iterates over Selector Sets.

If an error occurs in cfidrPIGetSelectorSets(), the error output argument is set and a valid Iterator ID is returned. In this case, calling the cfidrIterNextSelectorSet() function returns the Null OID.

PARAMETERS

error (output) A pointer to the error returned if this function fails. The caller is responsible for allocating the memory used by this parameter.

ERROR CODES

CFIDR_INTERNAL_SYSTEM_ERROR:
a system error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The cfidrIterNextSelectorSet() function returns the OID of the next Selector Set known by the DR-PI.

If no Selector Sets are known by the PI, an error is not returned but the first call to cfidrIterNextSelectorSet() returns the Null OID.

If cfidrPIInit() has not been called before, this function will call it before proceeding.

REFERENCE

cfidrIterNextSelectorSet()
cfidrPIInit()

6.4.7 cfidrPIOpenSelectorSet

DECLARATION

cfidrSelectorSetIdT cfidrPIOpenSelectorSet(
cfidrStringT name,
cfidrErrorT *error)

DESCRIPTION

This function opens a SelectorSet specified via *name* and allows it to be updated.

RETURN VALUE

The return value is an OID of type **cfidrSelectorSetIdT** for the SelectorSet just opened. If an error occurs, a Null OID is returned.

PARAMETERS

name (input) A string representing the name of the SelectorSet to be opened.

error (output) The error returned if this function fails. The caller is responsible for allocating memory used by this parameter.

ERROR CODES

CFIDR_INVALID_NAME:
name parameter is not a valid string or is not a legal name.

CFIDR_OBJECT_NOT_FOUND:
no SelectorSet with the specified *name* exists.

CFIDR_OPEN_FOR_UPDATE_FAILED:
a SelectorSet with the specified *name* exists or is open already but it is not possible to update it.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occured.

PRE-CONDITIONS

If the SelectorSet does not exist, it is not created automatically.

If cfidrPIINIT() has not been called previously, it is called automatically before this function.

POST-CONDITIONS

If the SelectorSet has been previously opened but not purged in the current session, then opening it again returns the same OID. In this case, the error output argument should be set to CFIDR_NO_ERROR to denote this fact even thought it is not an error.

6.4.8 cfidrSelectorSetCreateLibrarySelector

DECLARATION

```
cfidrLibrarySelectorIdT
cfidrSelectorSetCreateLibrarySelector(
cfidrSelectorSetIdT owner,
cfidrStringT libraryName,
cfidrErrorT *error)
```

DESCRIPTION

Creates a LibrarySelector in the SelectorSet specified by *owner* and adds it to the end of the list in the *LibrarySelectors* attribute of *owner*. The LibrarySelector defines which Libs are used for view selection.

RETURN VALUE

The return value is an OID of type **cfidrLibrarySelectorIdT** for the LibrarySelector just created. If an error occurs a Null OID is returned.

PARAMETERS

owner (input) The OID of the SelectorSet owning the Selector to be added.

libraryName (input) The string representing the LibraryName of the LibrarySelector.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a SelectorSet.

CFIDR_INVALID_NAME:
the libraryName parameter is not a valid string or is not a legal name.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

6.4.9 cfidrSelectorSetCreateViewNameSelector

DECLARATION

```
cfidrViewNameSelectorIdT
cfidrSelectorSetCreateViewNameSelector(
cfidrSelectorSetIdT owner,
cfidrStringT libName,
cfidrStringT cellName,
cfidrStringT viewName,
cfidrErrorT *error)
```

DESCRIPTION

Creates a ViewNameSelector in the *owner* SelectorSet. This routine adds the newly created selector to the end of the list of ViewSelectors associated with the SelectorSet specified by *owner*.

This ViewNameSelector may then be used by **cfidrSelectorSetSelectView()** to select a particular view. Both the *libName* and *cellName* parameters are optional. They are used to specify whether a ViewSelector should be used for a particular Cell. If the *libName* and/or the *cellName* attributes are not Null strings in the newly created ViewNameSelector then they must match the Cell passed to **cfidrSelectorSetSelectView()** in order for this ViewNameSelector to be used.

RETURN VALUE

The return value is an OID of type **cfidrViewNameSelectorIdT** for the ViewNameSelector just created. If an error occurs a Null OID is returned.

PARAMETERS

owner (input) The OID of the SelectorSet owning the Selector to be added.

libName (input) A cfidrStringT representing the *Name* of a Lib to be used as a match for the function **cfidrSelectorSetSelectView()**.

cellName (input) A cfidrStringT representing the *Name* of a Cell to be used as a match for the function **cfidrSelectorSetSelectView()**.

viewName (input) A cfidrStringT representing the *ViewName* of a View to be used as a match for the function **cfidrSelectorSetSelectView()**.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a SelectorSet.

CFIDR_INVALID_NAME:
either the *libName*, *cellName*, or viewName parameter is not a valid string or is not a legal name.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred

PRE-CONDITIONS

If the *libName* is a Null string, it acts like a wildcard and matches any Lib's *Name*.

If the *cellName* is a Null string, it acts like a wildcard and matches any Cell's *Name*.

POST-CONDITIONS

This ViewNameSelector cannot have its attributes changed later. If incorrect attributes are present, the application writer must destroy the Selector then create new selector(s) with the new attributes(s).

REFERENCE

cfidrSelectorSetSelectView()
cfidrObjectDestroy()

6.4.10 cfidrSelectorSetCreateViewTypeSelector

DECLARATION

cfidrViewTypeSelectorIdT
cfidrSelectorSetCreateViewTypeSelector(
cfidrSelectorSetIdT owner,
cfidrStringT libName,
cfidrStringT cellName,
cfidrStringT viewType,
cfidrErrorT *error)

DESCRIPTION

Creates a ViewTypeSelector in the *owner* SelectorSet. This routine adds the newly created selector to the end of the list of ViewSelectors associated with the SelectorSet specified by owner.

This ViewTypeSelector may then be used by **cfidrSelectorSetSelectView()** to select a particular view. Both the *libName* and *cellName* parameters are optional. They are used to specify whether a ViewSelector should be used for a particular Cell. If the *libName* and/or the *cellName* attributes are not Null strings in the newly created ViewTypeSelector then they must match the Cell passed to **cfidrSelectorSetSelectView()** in order for this ViewTypeSelector to be used.

RETURN VALUE

The return value is in OID of type **cfidrViewTypeSelectorIdT** for the ViewTypeSelector just created. If an error occurs a Null OID is returned.

PARAMETERS

owner (input) The OID of the SelectorSet owning the Selector to be added.

libName (input) A cfidrStringT representing the *Name* of a Lib to be used as a match for the function **cfidrSelectorSetSelectView()**.

cellName (input) A cfidrStringT representing the *Name* of a Cell to be used as a match for the function **cfidrSelectorSetSelectView()**.

viewType (input) A cfidrStringT representing the *ViewType* of a View to be used as a match for the function **cfidrSelectorSetSelectView()**.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
owner is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
owner is not the OID of a SelectorSet.

CFIDR_INVALID_NAME:
either the *libName* or *cellName* parameter is not a legal name, or the viewType parameter is not a valid

string.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred

PRE-CONDITIONS

If the *libName* is a Null string, it acts like a wildcard and matches any Lib's *Name*.

If the *cellName* is a Null string, it acts like a wildcard and matches any Cell's *Name*.

POST-CONDITIONS

This ViewTypeSelector cannot have its attributes changed later. If incorrect attributes are present, the application writer must destroy the entire Selector then create new selector(s) with the new attributes(s).

REFERENCE

cfidrSelectorSetSelectView()
cfidrObjectDestroy()

6.4.11 cfidrSelectorSetGetLibrarySelectors

DECLARATION

```
cfidrLibrarySelectorsIdT
cfidrSelectorSetGetLibrarySelectors(
cfidrSelectorSetIdT selectorSet,
cfidrErrorT *error)
```

DESCRIPTION

This function initiates traversal of all the LibrarySelectors in the LibrarySelectors list in the SelectorSet specified by *selectorSet*.

RETURN VALUE

The return value is a **cfidrLibrarySelectorsIdT** referring to an Iterator ID that iterates over LibrarySelectors. If an error occurs, a Null OID is returned, but the first call to cfidrIterNextLibrarySelector() returns a Null OID.

PARAMETERS

selectorSet (input) The OID of the *selectorSet* for which the Selectors relation is to be traversed.

error (output) The error returned if this function fails. The caller is responsible for allocating the

memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
selectorSet is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selectorSet is not the OID of a SelectorSet.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

POST-CONDITIONS

The **cfidrIterNextLibrarySelector()** function returns the OID of the next LibrarySelector known by the LibrarySelector relationship.

REFERENCE

cfidrIterNextLibrarySelector()

6.4.12 cfidrSelectorSetGetViewSelectors

DECLARATION

cfidrViewSelectorsIdT cfidrSelectorSetGetViewSelectors(
cfidrSelectorSetIdT selectorSet,
cfidrErrorT *error)

DESCRIPTION

This function initiates traversal of all the ViewSelectors at the end of the ViewSelectors relationship defined for the SelectorSet specified by *selectorSet*.

RETURN VALUE

The return value is a **cfidrViewSelectorsIdT** referring to an Iterator ID that iterates over ViewSelectors. If an error occurs, a valid Iterator is returned, but the first call to cfidrIterNextViewSelector() returns a Null OID.

PARAMETERS

selectorSet (input) The OID of the *selectorSet* for which the Selectors relation is to be traversed.

error (output) The error returned if this function fails. The caller is responsible for allocating the

memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
selectorSet is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selectorSet is not the OID of a SelectorSet.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrIterNextViewSelector()

6.4.13 cfidrSelectorSetPurge

DECLARATION

```
cfidrVoidT cfidrSelectorSetPurge(
cfidrSelectorSetIdT selectorSet,
cfidrErrorT *error);
```

DESCRIPTION

This function purges the SelectorSet specified by *selectorSet*. The effect of purging a SelectorSet is also to purge the entire Object hierarchy (all the selectors it contains) owned by the purged SelectorSet.

PARAMETERS

selectorSet (input) The OID of the SelectorSet to purge. All ViewSelectors and LibrarySelectors in the SelectorSet are also purged.

error (output) The error returned if this function fails. The caller is responsible for allocating memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
selectorSet is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selectorSet is not the OID of a SelectorSet.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

PRE-CONDITIONS

A SelectorSet can be purged regardless of the access mode of any object.

POST-CONDITIONS

Any OIDs referring to purged SelectorSets are made unusable. They may be reused to refer to other SelectorSets at some point, so the application cannot assume that if the purged SelectorSet is again made available (via, open, create, etc.) that it will have the same OID.

Policy: The purge operation removes any changes made to an object since the last save or open. The object then becomes unavailable and its OID becomes unusable. Opening the SelectorSet again assigns the SelectorSet an OID and makes it available for updating.

An example implementation that provides this behavior would make an in-memory copy of the SelectorSet when it was opened. Any changes to the SelectorSet, or to any of its contained objects, would be made to the in-memory copy of the SelectorSet. The effect of a "purge" is to delete the in-memory copy of the SelectorSet, leaving the on-disk persistent SelectorSet data unchanged.

Policy: Similar to open, purge also recursively walks the ownership hierarchy rooted by the specified SelectorSet and purges all Objects (LibrarySelectors and ViewSelectors) in this hierarchy. Unlike open, purge continues walking the ownership hierarchy when it encounters an Object which can be purged individually. Thus purge behaves similarly to save in this respect.

REFERENCE

cfidrSelectorSetSave()

6.4.14 cfidrSelectorSetSave

DECLARATION

```
cfidrVoidT cfidrSelectorSetSave(
cfidrSelectorSetIdT selectorSet,
cfidrErrorT *error)
```

DESCRIPTION

This function saves the SelectorSet specified via *selectorSet*. The effect of saving a SelectorSet is also to save the entire Object hierarchy (all selectors it contains) owned by the saved *selectorSet*. Saving a SelectorSet makes all updates to objects (selectors) in the SelectorSet's hierarchy persistent.

PARAMETERS

selectorSet (input) The OID of the SelectorSet to save. The ViewSelectors and LibrarySelectors of the *selectorSet* are also saved.

error (output) The *error* returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
selectorSet is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selectorSet is not the OID of a SelectorSet.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

6.4.15 cfidrSelectorSetSelectView

DECLARATION

```
cfidrViewIdT cfidrSelectorSetSelectView(
cfidrSelectorSetIdT selectorSet,
cfidrCellIdT cell,
cfidrErrorT *error)
```

DESCRIPTION

Uses the information in a Cell and a SelectorSet to compute the selected view. This routine uses the LibrarySelectors in the SelectorSet to determine which libraries contain views for selection and ViewSelectors to determine which selector should be selected.

This routine looks for the view in the libraries specified by the LibrarySelectors. cfidrSelectorSetSelectView() starts at the first ViewSelector and determines whether it selects a particular view. It continues through the ViewSelectors in order of creation until it finds a Selector which selects exactly one view. When cfidrSelectorSetSelectView() finds a ViewSelector that specifies a particular view, it returns that view to the calling program. If multiple views are selected by a ViewSelector the error CFIDR_MULTIPLE_VIEWS_MATCH is set and a Null OID is returned.

The ViewSelector specifies the selection of a particular ViewName or ViewType in the libraries specified by the LibrarySelector.

The following pseudo-code illustrates a possible algorithm for this routine. It is intended to demonstrate functionality rather than require this particular algorithm.

```
viewSelectorIterator =
```

```
  cfidrSelectorSetGetViewSelectors(selectorSet, &error);
viewSelector =

  cfidrIterNextViewSelector(viewSelectorIterator, &error);
error = CFIDR_NO_VIEW_MATCHES;

// *** iterate through the ViewSelectors to choose a View ***

while (cfidrObjectIsUsable(viewselector) == CFIDR_TRUE)

{

  // *** if we should look at this selector

  if ((viewSelector's LibName is null or

    matches cells LibName) AND

    (viewSelector's CellName is null or

    matches cells CellName))

  {

    // look at libraries in SelectorSet

    libraryIterator =

      cfidrSelectorSetGetLibrarySelectors(

      selectorSet, &error);

    librarySelector = cfidrIterNextLibrarySelector(

      libraryIterator, &error);

    while (cfidrObjectIsUsable(libraryselector,

      &error)  ==CFIDR_TRUE)

    {

      if (exactly 1 view exists in

         Lib/CellName with specified name/type)

      {

         error = CFIDR_NO_ERROR

       return view

      }

      if (>1 view found)

      {

       error = CFIDR_MULTIPLE_VIEWS_MATCH;
```

```
        return(cfidrPIGetNullId());
      }
    }
  }
}
// *** no object found
return(cfidrPIGetNullId());
```

RETURN VALUE

The return value is an OID of type **cfidrViewIdT** for the View selected. If an error occurs, a Null OID is returned.

PARAMETERS

selectorSet (input) The OID of the SelectorSet used to do view selection.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
selectorSet is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selectorSet is not the OID of a SelectorSet.

CFIDR_NO_VIEW_MATCHES:
No matches were found for views meeting the view selection criteria specified by the *selectorSet*.

CFIDR_MULTIPLE_VIEWS_MATCH:
If a ViewTypeSelector finds multiple views of the same type in the same cell owned by library then this error will be returned.

CFIDR_UNUSABLE_CELL_OID:
cell is not a usable OID.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

6.4.16 cfidrViewNameSelectorGetCellName

DECLARATION

cfidrStringT cfidrViewNameSelectorGetCellName(
cfidrViewNameSelectorIdT selector,
cfidrErrorT *error)

DESCRIPTION

Gets the *CellName* attribute of a ViewNameSelector

RETURN VALUE

The *CellName* of the ViewNameSelector. This may legally be a Null string if **cfidrSelectorSetCreateViewNameSelector()** was called with a Null *CellName* string. A Null string is also returned if there is an error, therefore the value at *error* must be checked whenever a Null string is returned to determine if the Null string is the correct return or if an error has occurred.

PARAMETERS

selector (input) The OID of the ViewNameSelector whose *CellName* attribute is desired.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
selector is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selector is not the OID of a ViewNameSelector.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred

REFERENCE

cfidrSelectorSetCreateViewNameSelector()

6.4.17 cfidrViewNameSelectorGetLibName

DECLARATION

cfidrStringT cfidrViewNameSelectorGetLibName(
cfidrViewNameSelectorIdT selector,
cfidrErrorT *error)

DESCRIPTION

Gets the *LibName* attribute of a ViewNameSelector

RETURN VALUE

The *LibName* of the ViewNameSelector. This may legally be a Null string, if **cfidrSelectorSetCreateViewNameSelector()** was called with a Null *LibName* string. A Null string is also returned if there is an error, therefore the value at *error* must be checked whenever a Null string is returned to determine if the Null string is the correct return or if an error has occurred.

PARAMETERS

selector (input) The OID of the ViewNameSelector whose *LibName* attribute is desired.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
selector is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selector is not the OID of a SelectorSet.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrSelectorSetCreateViewNameSelector()

6.4.18 cfidrViewNameSelectorGetViewName

DECLARATION

cfidrStringT cfidrViewNameSelectorGetViewName(
cfidrViewNameSelectorIdT selector,
cfidrErrorT *error)

DESCRIPTION

Gets the *ViewName* attribute of a ViewNameSelector

RETURN VALUE

The *ViewName* of the ViewNameSelector. This may legally be a Null string, if **cfidrSelectorSetCreateViewNameSelector()** was called with a Null *ViewName* string. It will be a Null string if there is an error. If there is an error a Null string is also returned, therefore the value at *error* must be checked whenever a Null string is returned to determine if the Null string is the correct return or if an error has occurred.

PARAMETERS

selector (input) The OID of the ViewNameSelector whose *ViewName* attribute is desired.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

RETURN VALUE

CFIDR_UNUSABLE_OID:
selector is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selector is not the OID of a ViewNameSelector.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrSelectorSetCreateViewNameSelector()

6.4.19 cfidrViewTypeSelectorGetCellName

DECLARATION

```
cfidrStringT cfidrViewTypeSelectorGetCellName(
cfidrViewTypeSelectorIdT selector,
cfidrErrorT *error)
```

DESCRIPTION

Gets the *CellName* attribute of a ViewTypeSelector.

RETURN VALUE

The *CellName* of the ViewTypeSelector. This may legally be a Null string if **cfidrSelectorSetCreateViewTypeSelector()** was called with a Null *CellName* string. If there is an error a Null string is also returned, therefore the value at *error* must be checked whenever a Null string is returned to determine if the Null string is the correct return or if an error has occurred.

PARAMETERS

selector (input) The OID of the ViewTypeSelector whose *CellName* attribute is desired.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
selector is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selector is not the OID of a ViewTypeSelector.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrSelectorSetCreateViewTypeSelector()

6.4.20 cfidrViewTypeSelectorGetLibName

DECLARATION

```
cfidrStringT cfidrViewTypeSelectorGetLibName(
cfidrViewTypeSelectorIdT selector,
cfidrErrorT *error)
```

DESCRIPTION

Gets the *LibName* attribute of a ViewTypeSelector.

RETURN VALUE

The *LibName* of the ViewTypeSelector. This may legally be a Null string if **cfidrSelectorSetCreateViewTypeSelector()** was called with a Null *LibName* string. If there is an error a Null string is also returned, therefore the value at *error* must be checked whenever a Null string is returned to determine if the Null string is the correct return or if an error has occurred.

PARAMETERS

selector (input) The OID of the ViewTypeSelector whose *LibName* attribute is desired.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
selector is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selector is not the OID of a ViewTypeSelector.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrSelectorSetCreateViewTypeSelector()

6.4.21 cfidrViewTypeSelectorGetViewType

DECLARATION

```
cfidrStringT cfidrViewTypeSelectorGetViewType(
cfidrViewTypeSelectorSetIdT selector,
cfidrErrorT *error)
```

DESCRIPTION

Gets the *ViewType* attribute of a ViewTypeSelector.

RETURN VALUE

The *ViewType* of the ViewTypeSelector. It will be a Null string if there is an error.

PARAMETERS

selector (input) The OID of the ViewTypeSelector whose ViewType attribute is desired.

error (output) The error returned if this function fails. The caller is responsible for allocating the memory for this parameter.

ERROR CODES

CFIDR_UNUSABLE_OID:
selector is not a usable OID.

CFIDR_INVALID_OBJECTTYPE:
selector is not the OID of a ViewTypeSelector.

CFIDR_INTERNAL_SYSTEM_ERROR:
some other error occurred.

CFIDR_NO_ERROR:
no error occurred.

REFERENCE

cfidrSelectorSetCreateViewTypeSelector()

Table of Contents Next Chapter

7 References

[1] *EXPRESS Language Reference Manual* , ISO TC184/SC4/WG5 Document N14, March 29, 1991.
[2] *Information Model and (example) PI*, Version 0.2, CFI Document 118, August 17, 1990.
[3] *EIS ECAD Domain Model*, Version 2.0, CFI Document 120, September, 1990.
[4] Loomis, Mary E. S., *The Database Book*, Macmillan Publishing Company, 1987.
[5] Electronic Design Interchange Format, Version 2 0 0. Paul Stanford and Paul Mancuso, Editors. Electronic Industries Association, Washington, D.C., 1989.